INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

New Mechanisms for Improved Adaptive Routing in Wormhole Networks

BY

Zaki Al-Awwami

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Zaki Al-Awwami** under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE**.

Thesis Committee

10/11/01

Dr. Sulaiman Al-Bassam (*Chairman*)

Dr. Mohammad Al-Mulhem (*Member*)

Dr. Mayez Al-Mouhamad (*Member*)

Kanaan Faisal 10/11/2001

Department Chairman, Prof. Kanaan A. Faisal

Dean of Graduate Studies, Prof. Osama A. Jannadi

20/11/2001

Date

عمادة الدراسات العليا

DEANSHIP OF GRADUATE STUDIES

New Mechanisms for Improved Adaptive Routing in Wormhole Networks

BY

Zaki Al-Awwami

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

In Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE
IN

COMPUTER SCIENCE

KING FAHD UNIVERSITY
OF PETROLEUM AND MINERALS
Dhahran, Saudi Arabia

October, 2001

DEDICATION

To my father, whose journey through life has demonstrated the true meaning of hard work, courage, and perseverance. I attribute and dedicate this work to your valuable and imprinted words for higher academic achievements.

May your soul rest in eternal peace.

ACKNOWLEDGEMENT

I would like to express my sincere appreciation for the thesis committee and the chairman for the encouragement and trust that they have extended to me. Many thanks to Dr. Sulaiman Al-Bassam, Dr. Mohammed Al-Mulhem, and Dr. Mayez Al-Mouhamed.

I would also like to express my deep gratitude for my main mentor in this research, Dr. Mohammad Obaidat for his guidance and generous support while he was at King Fahd University of Petroleum and Minerals and via numerous correspondences from the US. Dr. Obaidat was on the thesis committee, but unfortunately could not attend the thesis defense. I am particularly grateful and fortunate to have worked with him, and my thesis committee members, as I have gained valuable insights into this field and have acquired the necessary skills to publish this literature into various respected international conferences and journals.

I would also like to thank all my family and friends who cared to share this experience and provided mental support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THESIS ABSTRACT

Name: Zaki Al-Awwami

Title: New Mechanisms for Improved Adaptive Routing in Wormhole Networks.

Major Field: Computer Science

Date of Degree: October 2001

Massively Parallel Processors are looked upon as the forthcoming computer architecture that can deliver the next awaited leap in processing power. This leveraged computational platform is desperately in demand today for the sake of solving and concurring large and time consuming problems. The processing prowess of these multicomputers is achieved by connecting hundreds and thousands of processing units over a highly optimized interconnection network. In order for these processors to collectively manage a complex problem, they must be able to achieve extremely fast communication amongst themselves. The performance of the interconnecting network is therefore of paramount significance to the performance of the entire parallel machine. Wormhole switching is an efficient form of switching that has been successfully applied to the interconnection networks of multicomputers with the performance objective of the overall machine in focus. Deadlock is the most formidable obstacle that any routing algorithm must address and overcome. It has been established through research that deadlocks occur very infrequently and that their seldom occurrence takes place only in situations where the network is close to or has exceeded its saturation point. This observation has led to the wide acceptance of true fully adaptive routing algorithms as viable alternatives to other deadlock-prevention routing algorithms.

This thesis proposes using the true fully adaptive routing algorithm with a new deadlock-recovery mechanism. The deadlock-recovery mechanism is efficient and takes advantage of the concept of wormhole switching in terms of low hardware resource requirements. The performance of the new mechanism can match that of other more expensive deadlock-recovery mechanisms, while requiring less hardware resources that are not on the critical path of the switching process. The proposed mechanism creates a new category of deadlock recovery techniques that we refer to as preemptive as opposed to the existing progressive and regressive categories.

The thesis also proposes a new injection limitation mechanism to be used along the proposed deadlock-recovery mechanism. As networks approach or exceed their saturation point, the performance of the network may degrade due to traffic congestion. This situation is exacerbated by the tendency of the network to form cyclic dependencies at this stage of the network, which could eventually lead to more deadlock formations. The proposed injection limitation mechanism attempts to prevent the early saturation and the performance degradation behavior by sensing and controlling the level of traffic congestion in the network. The new mechanism outperforms other injection limitation mechanisms proposed in the literature. The performance evaluation of all the selected routing algorithms and proposed mechanisms in this thesis is achieved via simulation. A large stochastic object-oriented simulator was developed and is used for this purpose throughout the thesis.

MASTER OF SCIENCE DEGREE

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
Dhahran, Saudi Arabia

October 2001

خلاصة الرسالة

اسم الطالب الكامل : زكي حسين العوامي

عنوان الدراسة : نظم جديده لتطوير طرق التسيير المتكيفة في شبكات التنقل الدودي

التخصص : علوم الحاسب و المعلومات

تاريخ الشهادة : أكتوبر 2001م – شعبان 1422ه

يتم التطلع إلى أنظمة المعالجات كبيرة التعدد على أنها التصميم الحاسوبي المقبل و الذي باستطاعته تحقيق الخطوة المنتظرة في سرعة المعالجة. نحن في حاجة ماسة لهذه الإمكانية الحاسوبية المتفوقة لكي يتم تطويع و تخطي مشاكل كبيره و مستهلكة للوقت. يتم تحقيق قوة المعالجة في هذه الحاسبات المتعددة عن طريق توصيل المئات و الآلاف من حدات المعالجة ضمن شبكات مترابطة محسنة. من اجل قيام هذه المعالجات بالتعاون لحل مشكلة معقده, يجب عليهم تحقيق سرعة فائقة في التواصل فيما بينهم. أداء الشبكات المترابطة هو إذا ذي أهمية قصوى للأداء الكلي لهذه الحاسبات المتعددة. التنقل الدودي هو أحد طرق التنقل المتطورة التي تم تطبيقها بنجاح في الشبكات المترابطة للحاسبات المتعددة من أجل الوصول إلى الهدف الأداني المنشود. الإنغلاق اللاحركي هو من أهم العقبات التي يتوجب على أي نظام تسيير التطرق لها و معالجتها. لقد تم التوصل عن طريق البحث إلى أن الإنغلاق اللاحركي قلما يحدث ولأن ندرة حدوثه تكون فقط في تلك الحالات الذي تكون فيها الشبكة قريبه من أو تخطت نقطة التشبع. هذه الملاحظة أدت إلى التقبل الواسع لطرق التسيير كاملة و حقيقية التكيف كخيار مفضل بالمقارنة مع طرق التسيير المانعة للإنغلاق اللاحركي.
هذه الرسالة تقترح تطبيق طريق تسيير كامل و حقيقي التكيف مع نظام جديد للتغلب على الإنغلاق اللاحركي. هذا النظام الجديد ضد الإنغلاق اللاحركي متفوق و يستفيد من مبادئ التنقل الدودي من حيث إنخفاض المتطلبات المادية. أداء النظام الجديد يعادل نظم أخرى متغلبة للإنغلاق اللاحركي و أكثر تكلفة منه على الرغم من متطلباته المنخفضة التكلفة و التي لا تؤثر على المسار الحرج لمنظومة التنقل. النظام المقترح يخلق نوعيه جديده من طرق التغلب على الإنغلاق اللاحركي و الذي نسميه بالنوعية المفرغة بالمقارنة مع النوعيات الموجودة الأخرى و المصنفة بالتقدمية و الرجعية.
هذه الرسالة تقترح أيضا نظام جديد لتقنين الحقن من أجل الاستخدام بالتوافق مع النظام الجديد المتغلب للإنغلاق اللاحركي. عندما تقترب الشبكات من أو تتعدى نقاط التشبع, أداء هذه الشبكات قد يهبط بفعل إزدحام المسارات. هذه الحالة معرضة للتفاقم بسبب ميول الشبكات لتشكيل حلقات متداخلة في هذه المرحلة, مما قد يقود حتميا إلى تكوين المزيد من تشكيلات الإنغلاق اللاحركي. النظام المقترح من أجل تقنين الحقن يهدف إلى منع التشبع المبكر و الهبوط الأداني عن طريق التحسس و السيطرة على معدلات إزدحام المسارات في الشبكة. هذا النظام الجديد يتفوق على نظم تقنين الحقن الأخرى المقترحة في هذا المجال. التقييم الأداني لكل طرق التسيير المختارة و المقترحة في هذه الرسالة يتم تحقيقه عن طريق المحاكاة. قد تم تطوير واستخدام برنامج محاكاة رقمي من أجل هذا الغرض في جميع أجزاء هذه الرسالة.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول و المعادن
الظهران , المملكة العربية السعودية

أكتوبر 2001م – شعبان 1422ه

CHAPTER 1

INTRODUCTION

Massively parallel processors (MPP) commonly referred to as multicomputers are an important class of high performance parallel machines that will provide the envisioned computation platform for solving the so-called grand challenge problems. Several variations of these machines are commercially available today. Continuous research in this field is needed to refine and optimize the design of these machines for maximum and efficient performance. This chapter presents the motivation behind this effort. It clearly defines the problem being pursued, and provides a list of accomplishments that were devised in order to provide solutions for the stated problem in light of the mentioned motives.

1.1 Motivation

Parallel architectures are called upon in order to provide the next leap in performance magnitude needed to solve inherently complex and time-consuming problems. The required orders of magnitude cannot be achieved using single processor, nor small-scale multiprocessing systems. Multicomputers are an important class of highly parallel architecture machines. Multicomputers tackle a problem by decomposing and distributing it over many processing elements that must communicate with each other over an interconnection network in order to cooperatively solve the problem. It is therefore very important to devise reliable and efficient message passing mechanisms in these multicomputers so as to optimize the internal communication process. The routing technique employed in order to transfer messages amongst all the processing elements of the multicomputer is at the heart of the message passing architecture that defines these machines. Accordingly, improving and optimizing the message routing algorithms in multicomputer interconnection networks is an essential step toward achieving the stated objective of an effective high performance platform.

1.2 Problem

Adaptive routing algorithms are the preferred mechanism for exchanging packets amongst all the nodes in an interconnection network for various reasons that will be discussed in more details in the following chapters. Since the amount of adaptivity provided by the routing algorithm is generally correlated to the underlying router hardware complexity. This hardware complexity in turn has the undesirable consequence of reducing the clock speeds at which the network can operate. The additional hardware resources have to be justified by a comparable increase in router performance. Certain classes of adaptive routing algorithms do not meet this objective. In particular, the amount of resources dedicated to avoiding deadlocks in deadlock-avoidance adaptive routing algorithms tends to be high and is therefore a major contributor to the hardware complexity of the router. These extra hardware resources are used to ensure freedom of deadlock, and do not contribute to the performance of the router as they have the potential to do. Despite various attempts at curtailing the amount of adaptivity provided by these routing algorithms in order to reduce their hardware requirements, still the cost of resources dedicated to deadlock avoidance compared to the resources used to perform the partial adaptive routing may not produce efficient router designs. The class of True Fully Adaptive Routing algorithms, therefore, emerges as the preferred routing mechanism due to the fact that all the hardware resources deployed are fully utilized to route packets, and, therefore, directly contribute to the performance of the router.

Deadlock resolution is a critical issue that affects the design of these routing algorithms, and consequently the underlying router hardware complexity. Deadlocks are generally rare events, and are more likely to occur as networks reach or are beyond their saturation point [30, 31]. To this effect, almost all deadlock-avoidance algorithms allocate more expensive resources to avoid deadlock than is necessary. This observation has paved the way for deadlock-recovery techniques to become more widely accepted as viable alternatives to deadlock-avoidance algorithms.

True fully adaptive algorithms route packets on all available channel resources without guarding against the possibility of deadlock formations. These algorithms therefore require an accompanying deadlock recovery mechanism. When deadlocks occur, it is important to resolve and drain them quickly. The deadlock recovery mechanism, in accordance with the observation that deadlocks are rare events, should be as efficient as possible without allocating any more expensive hardware resources than is necessary.

In addition, wormhole routed networks generally do not achieve very high throughput, or high channel utilization, especially as the network reaches saturation. Therefore, they generally suffer from a low saturation point. As networks approach or exceed their saturation point, more cyclic dependencies start to form, which can lead to deadlock [30, 31]. The constant injection rate of packets will exacerbate the problem further and lead to more deadlock situations, which will have further adverse effects on the throughput and stability of the network.

1.3 Achievements

From the previous section two characteristics of the proposed improved adaptive routing algorithm become apparent. First the routing algorithm should be highly adaptive with a deadlock-recovery mechanism, which would allow most of the resources to be used to enhance the adaptivity of the algorithm. Second, the router should employ an injection limitation, or throttle, mechanism in order to avoid the degraded throughput when the saturation point of the network has been reached or exceeded, and hopefully generally improve the performance of the algorithm as well. The following sections list the achievements in more detail.

1.3.1 True Fully Adaptive Routing Algorithm

The routing algorithm proposed is a true fully adaptive routing (TFAR) algorithm. As mentioned earlier the TFAR algorithm allows adaptive routing on all available channel and virtual lane resources, without protecting against the possibility of deadlock formations.

1.3.2 Deadlock Recovery Mechanism

The TFAR algorithm requires a deadlock recovery mechanism. The proposed deadlock recovery mechanism is a novel approach. Since deadlocks are rare events, only as few resources as possible should be dedicated to handle these rare occurrences. The proposed approach takes advantage of the concept behind wormhole routing. Namely, the low number of edge buffers requirements per channel, which can be as low as one flit buffer, and the flow control mechanism already in place, and which carries control information in the opposite direction to that of data flow. The proposed mechanism creates a new category of deadlock-recovery techniques. Previously known categories are either progressive or regressive. We refer to this new category as a *preemptive* deadlock recovery.

1.3.3 Injection Limitation Mechanism

A new injection limitation or throttle mechanism is also introduced as part of this thesis. The mechanism attempts to remedy the low throughput problem of wormhole switched networks. It serves two functions. First it protects against the degrading effect of reaching the saturation point on the performance of the network. Second, it reduces the probability of forming cyclic dependencies amongst packets, which can eventually lead to a reduction in the number of deadlock cycles. As mentioned earlier, cyclic dependencies are more likely to form when the network is near or beyond its saturation point.

The injection limitation mechanism simply reduces the rate of packet injection in response to a high congestion level detected on a particular channel of the network. The mechanism is dynamic in relation to traffic load rates, and traffic patterns. It is based only on locally computed information at each node for performance reasons.

1.3.4 Discrete-event Simulator

Also as part of this effort, a discrete-event, stochastic, flit-level simulator of wormhole switched interconnection networks has been designed and built. The simulator named *WormSim* is a 5000 Lines of Code (LOC) written in Java. It will be used to test, tune, and evaluate the performance of the proposed mechanisms against the various selected routing algorithms from the literature. It was used to obtain all the performance results shown throughout this thesis.

The simulator was created in Java, a true object-oriented language, so that it can be hierarchical, modular, extensible, flexible, and reusable. The simulator is capable of modeling any k-ary n-cube topology interconnection network. It is very flexible and can be used to evaluate the performance of various routing algorithms, selection functions, traffic distributions, and network configurations.

1.4 Thesis Organization

The remaining thesis chapters are organized as follows. Chapter 2 provides detailed information on wormhole switched networks. It unveils various properties of interconnection networks, and provides a survey of various switching techniques used leading to wormhole switching. Chapter 3 provides an in depth classification and survey of various routing algorithms and their design methodologies. Chapter 4 sets the rules for the performance evaluation effort. It provides all the definitions and assumptions made about the network model, message model, and performance metrics collected. Chapter 5 introduces the

proposed deadlock-recovery mechanism, its operation details and simulation results. Chapter 6 introduces the proposed injection limitation mechanism, its operation details, and simulation results as well. A conclusion and future work considerations in this area of research is contemplated in chapter 7.

CHAPTER 2

BACKGROUND: WORMHOLE SWTICHED NETWORKS

Message-passing multicomputers are composed of many nodes that communicate with each other over a set of switches and communication links collectively known as the interconnection network. The performance of the interconnection network is therefore the most critical factor affecting the performance of the entire parallel machine [1, 2]. Wormhole switching is the most popular switching technique that has been applied to the interconnection networks of parallel multicomputers. It is well suited for application in message-passing networks as it allows for the design of simple, fast, and low-cost hardware router nodes while providing low latency, high-bandwidth communication [3].

In the following sections, some background information about interconnection networks and their classifications into direct and indirect networks are provided. The main advantages of both of these classes and their areas of applications are highlighted. Emphasis and more details are provided for direct networks and their characteristics, such as topology, switching technique, flow control, and routing. Finally, definitions of various important concepts that pertain to wormhole switched interconnection networks are discussed.

2.1 Interconnection Networks

Interconnection networks were originally developed and used for message-passing multicomputers. The success of these interconnects, mainly due to their scalability and performance characteristics caused the technology and the term behind it to be transferred and applied to many other fields such as distributed shared-memory multiprocessors. Today the technology is being applied to a wide spectrum of applications such as internal networks for ATM (Asynchronous Transfer Mode) switches, NOWs

(Network of Workstations), SANs (System Area Networks), LANs (Local Area Networks), MANs (Metropolitan Area Networks), WANs (Wide Area Networks), telephone switches, backplane buses, processor-to-memory interconnects in supercomputer class machines, and recently to NAS (Network Attached Storage) specialized servers [4].

Interconnection networks play a critical role in the performance of the entire parallel machine. They have various design criteria of varying degree of importance when applied to different environments. Some of these criteria are performance requirements, scalability, incremental scalability, partitionability, simplicity, distance span, physical limitations, reliability, workload patterns, and finally cost constraints [4].

There are two major types of interconnection networks, direct and indirect interconnection networks, which are covered in the following sections.

2.1.1 Direct Networks

The interconnection network is called *direct* if each node is connected directly to a switch. The nodes communicate with each other via their switches, which are connected by a set of point-to-point links. The switches in this case are commonly referred to as routers. In other words, direct interconnection networks embed the router and processing nodes together inside the topology of the network. The most common direct network topologies are the class of k-ary n-cube networks, which encompass rings, meshes, and tori.

Most commercial multicomputer implementations utilized direct networks. The Intel Paragon used a 2-D mesh, while the MIT J-Machine used a 3-D mesh. KSR (Kendall Square Research), the Intel/CMU iWarp, and the Cray T3D and T3E all utilized a 1-D Unidirectional Torus, a 2-D Bidirectional Torus, and a 3-D Bidirectional Torus respectively. The Intel iPSC and nCUBE machines both utilized a hypercube. [4].

Many other direct network topologies have been proposed and studied in the literature for use in parallel multicomputers such as stars, rotator graphs, de Bruijn networks, express cubes, packed

exponential connections (PEC), and binary n-cubes. The binary *n*-cubes, which are special cases of the family of *k*-ary *n*-cube networks, are cubes with *n* dimensions and *k* nodes in each dimension [3, 5].

2.1.2 Indirect Networks

In *indirect networks*, the processing nodes are not integrated with the switches. The interconnection network is composed of many switches. The processing nodes are only connected to a subset of those switches that lie on the edges of the interconnection network, and hence can only communicate with each other by traversing one or more of the intermediate connecting switches. This is the reason why these networks are also known as switch-based networks. Indirect or switched-based networks can be either regular or irregular. Regular topologies conform to a regular switch connection pattern, while irregular topologies do not.

Multistage Interconnection Networks (MIN) are a major class of indirect regular interconnection network topologies. They are composed of a number of switch stages; each switch employs a crossbar switching fabric. The processors and devices are connected to the first and last stages of the network. There are many variations of MINs and are mainly classified by the number of stages they contain, and the connection patterns used to connect the stages together. These two characteristics of MINs determine the total capacity, and the routing flexibility of the interconnection network. MINs were initially proposed for telephone switch networks. Commonly used indirect networks are generally referred to as MINs, BMINs (Bidirectional MINs), and fat-trees. More specific examples of MINs are Beneš Networks, Multistage Baseline Networks, Multistage Butterfly Networks, and Omega Networks [1, 4, 5, 6].

Regular indirect networks are mostly used for constructing what is known as *scalable parallel computers* (SPC). Some commercial implementations that utilize these networks are the Cray X/Y-MP with a crossbar implementation. IBM SP-1/2, CM-5, and Meiko CS-2 utilize Bidirectional MINs [1, 4].

Irregular indirect network topologies evolved out of the need to construct inexpensive parallel computer networks on top of existing LANs and workstations. This was mainly driven by the success of

interconnection networks and their switching mechanisms in terms of their high performance and low cost in comparison to high-speed LAN backbones such as ATM. Also building a custom interconnection network for this purpose can be expensive. Implementing this technology for LANs has produced Gigabit speed networks. Irregular networks are similar to regular networks, but do not have a regular pattern of where processors and workstations are attached into the network. This is due to the nature of the geographically dispersed workstation connections in a LAN or WAN. Most characteristics that apply to other network topologies, such as switching, flow control, and routing apply to irregular networks as well with minor considerations [7, 8]. Examples of Irregular indirect networks are the NOWs, Myrinet, DEC Autonet, ATOMIC, and ServerNet [9, 10, 11, 12, 13].

Research in the interconnection network area has mainly focused on direct interconnection networks for many reasons. Direct networks **scale uniformly**. When the number of nodes increase, so does the total communication bandwidth, memory bandwidth, and the total processing power of the entire parallel machine [14]. Another important feature of direct networks is their ability to exploit **communication locality** of parallel applications. The significance of this feature stems from the fact that local packet exchanges not only traverses fewer hops, but also utilizes only a small fraction of the total network bandwidth. Although Indirect networks have generally lower latency, higher bandwidth, and simpler deadlock prevention mechanisms, it has been shown in [6] that communication locality can make low-dimensional direct networks perform comparably, or even favorably to indirect networks. Direct networks have become the most popular architecture to implement massively parallel processors. These multicomputers will not only scale well to a large number of nodes, but will continue to take advantage of communication locality as they grow [14]. Also as VLSI systems are known to be wire-limited, a study has concluded that high-bandwidth direct networks are preferable to indirect networks on account of the communication locality effect [3]. This is not to say that direct networks are clear winners in all cases. Another study compares the two alternatives on the basis of modularity, incremental scalability, partitionability, simplicity, and pin-out constraints. It concludes that neither topology is clearly superior for all the above-mentioned criteria; rather two of these topologies rise

above the others, namely *k*-ary *n*-cubes and BMINs [5]. This effort will accordingly only consider direct networks, and more specifically the *k*-ary *n*-cube topologies for the remainder of this thesis.

A direct network is characterized mainly by its *topology*, *switching technique*, *flow control*, and *routing* [14]. The following sections will elaborate more on these characteristics.

2.2 Topology

A direct interconnection network has channels that directly connect each node to only a few other nodes, called neighbors. A node communicates with another node that is not its neighbor by passing the message to one of its neighbors for forwarding. Neighbors of a node are usually defined by the topology of the network.

More formally *topology* refers to an interconnection network that is a strongly connected graph, $I = G(N, C)$. The vertices of I are a set of nodes, N. The edges are a set of channels, $C \subseteq N \times N$. Each channel is unidirectional and carries data from a source node to a destination node. A bidirectional network is one where [15]:

$$(n1, n2) \in C \Rightarrow (n2, n1) \in C$$

A *k*-ary *n*-cube topology is a radix *k* cube with *n* dimensions. The radix indicates that there are *k* nodes in each dimension for a total of $N = k^n$ nodes. The simplest way to arrange *n* processors in a network is a one-dimensional array ($n = 1$). If processors are numbered 1,2...*n*, then processor *i* is linked to its two neighbors *i*-1 and *i*+1 through a two-way communication link. End processors 1 and *n* each have only one neighbor. Figure 2.1a shows a linear array of *n* processor [16]. If the end nodes in the linear array are connected then we get a ring or a torus network, as shown in Figure 2.1b. More generally for bidirectional cubes, the end-channels are not essential, if they are present, the cube is called torus connected, and if not it is called mesh connected. Most parallel computers have been

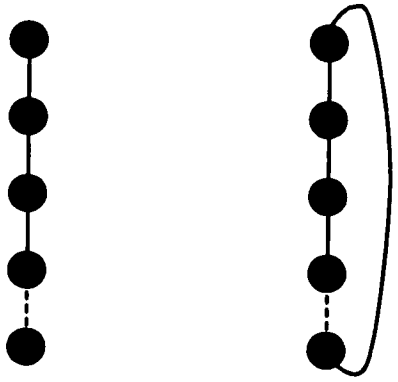
constructed using networks that are either k-ary n-cubes or are isomorphic to k-ary n-cubes such as rings, meshes, and tori [3]. There are three basic types of k-ary n-cubes:

1) Unidirectional,

2) Torus-connected bidirectional, and

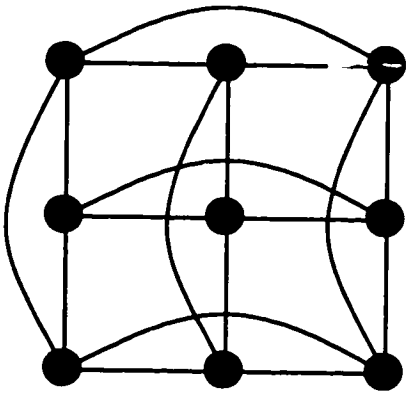
3) Mesh-connected bidirectional.

A k-ary n-cube is constructed from k k-ary $(n-1)$-cubes by connecting like elements into rings or linear arrays. Each node has an address that is an n digit, radix k number, $a_{n-1},\ldots a_0$. Each address digit, a_d, represents the coordinate of a node in dimension d and can take on values in the range of [0, k-1]. In a torus network, nodes are connected to all other nodes with an address that differs in only one digit by ± 1 mod k. In a mesh, nodes are connected to all other nodes with an address that differs in one digit by ± 1 where the result is in the range of $[0, k-1]$ [15, 17]. Figures 2.1c, and 2.1d illustrate a 3-ary 2-cube torus, and a 3 x 3 x 3 3D mesh respectively. A hypercube is a symmetric network and a special case of both the n-dimensional mesh and the k-ary n-cube networks. The hypercube can be considered as an n-dimensional mesh in which $k_i = 2$ for all i, $0 < i \leq n-1$, or a 2-ary n-cube. Figure 2.1e illustrates a hypercube [14].

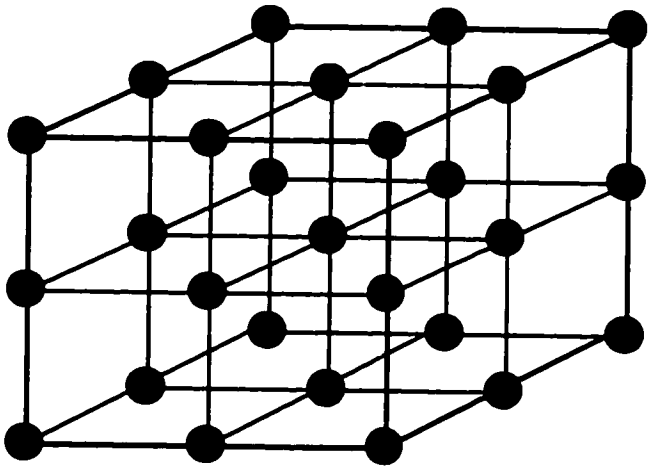


(a) Linear array with n processors

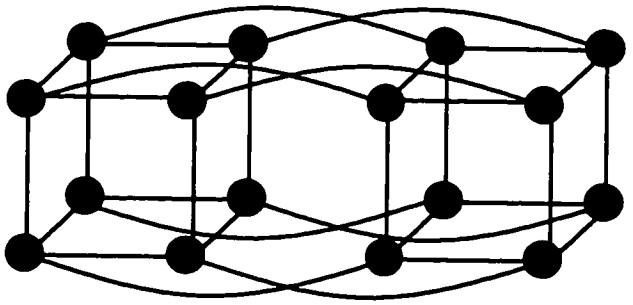
(b) 1-Dimensional Ring



(c) 3-ary 2-cube Torus



(d) 3 x 3 x 3 3D Mesh



(e) Hypercube

Figure 2.1 Direct network topology examples.

2.2.1 Topology Characteristics

Various important definitions that are used to further characterize and evaluate the topology of direct networks are listed below: [4, 14, 18].

- **Bisection Width (*BW*):** The minimum number of channels that must be removed in order to separate the network into two equal halves (this is basically the partition problem).
- **Channel Width (*CW*):** The number of bits a physical channel can simultaneously transmit between two neighboring nodes.
- **Channel Rate (*CR*):** The maximum rate of bits that can be transferred by the physical channel.
- **Channel Bandwidth (*CB*):** The product of the channel width and the channel rate: $CB = CW \times CR$. Channel bandwidth is used as a performance measure of the communication capability of the direct network.
- **Bisection Density (*BD*):** The product of the bisection width and the channel width: $BD = BW \times CW$. It is used as an indication of network cost. This means that the higher the bisection width, the lower the channel width is for a given bisection density.
- **Degree:** The degree of a node d, is the number of channels incident to the node. The in-channels are called *in-degree*, and the out-channels are called *out-degree*. The total degree is the sum of those two numbers. The degree of a node directly affects the number of pins per node, the control logic of the router, and therefore the total cost of the node.
- **Regularity:** A network is called *regular* when all its nodes have the same degree. The mesh network is therefore not regular, while the torus and hypercube networks are regular.
- **Diameter:** The maximum distance between two nodes in the network. It is used as the primary figure of merit for a network, the shorter the diameter the better the performance.
- **Wire Length:** The length of the wires in the network. It determines the speed at which the network can operate and the amount of power dissipated by driving these wires.

- **Symmetry:** A symmetric network looks the same when any node is chosen as the origin. In this case the network is called isomorphic to itself. Rings, tori, and hypercubes are symmetric, while linear arrays and meshes are not. Symmetric networks have important characteristics, which simplify their management.
- **Orthogonality:** A network topology is orthogonal if and only if nodes can be arranged in an orthogonal *n*-dimensional space, and each link can be arranged such that it produces a displacement in one of the dimensions of the network. Orthogonal networks can be further classified into strictly orthogonal, and weakly orthogonal. If the nodes in the network have at least one link in each of the network dimensions, then the topology is *strictly orthogonal*. Where in *weakly orthogonal* networks, some nodes may not have a link in one or more dimensions. Strictly orthogonal network topologies are very important as they simplify the routing decision so that it can be implemented efficiently in hardware. *K*-ary *n*-cube networks, meshes, tori, hypercubes, and n-dimensional networks are all examples of strictly orthogonal network topologies.

Examples of machines using direct networks include the Caltech Cosmic Cube, and the Connection Machine. Within the direct network class of networks, low-dimensional networks are favored for various reasons. They scale better than high-dimensional networks, they are modular, and are easy to implement. They have lower bisection widths, and consequently they can offer wider channels and higher channel bandwidth for a given bisection density. Dally in [3] has analyzed the performance of *k*-ary *n*-cube networks, and has shown that low-dimensional networks achieve lower latency and better hot-spot throughput than high-dimensional networks. Although low-dimensional networks have relatively large average distance between nodes, they are still popular for use in wormhole switching systems. In wormhole routing, the path length has minimal effect on network latency, which eliminates any undesired impact on the performance of these systems [6, 14].

2.3 Switching Techniques

The switching mechanism handles the actual passage of data from the input channels to the output channels, through the switching fabric. The switching technique used within the switch directly influences the network latency [6]. A variety of switching techniques have been used in interconnection networks as a result of continuous development and enhancements efforts. Wormhole switching has long been the switching technique of choice for direct networks. Following sections discuss the various switching techniques in more detail. An equation that describes the basic no-load latency using each switching technique is provided. This latency assumes that there is no contention due to the existence of other packets in the network. The message size is assumed to be L-bits long. The flit size is W bits wide. Channel width is of the same size as a flit, so it can be transferred in a single time unit. Packet size is therefore $L+W$ bits. The physical channel can operate at F Hz between adjacent routers. The propagation delay, t_w, across the channel is therefore:

$$t_w = \frac{1}{F}$$

Channel bandwidth is CB bits per second. The time it takes the router to process the header of a packet is t_r, while the switching delay through the router is denoted by t_s. The distance between source and destination nodes is D hops.

2.3.1 Circuit Switching

In *circuit switching*, a dedicated circuit must be established between any two nodes, prior to any communication between them can take place. If a node wants to communicate to another node, it first

sends a control message toward that node reserving the path as it advances along. An acknowledgment is returned to the originating node if the attempt was successful. At this point communication can take place between the two nodes using the reserved channel. The channel is then specifically torn down at the end of communication for reuse by other nodes. Figure 2.2 below provides an illustration of circuit switching. The figure shows a time diagram of the operation of circuit switching. The vertical axis represents nodes that a packet travels through along the path to its destination, starting from a source node (*S*), and going through intermediate nodes (*I*1, *I*2, and *I*3):

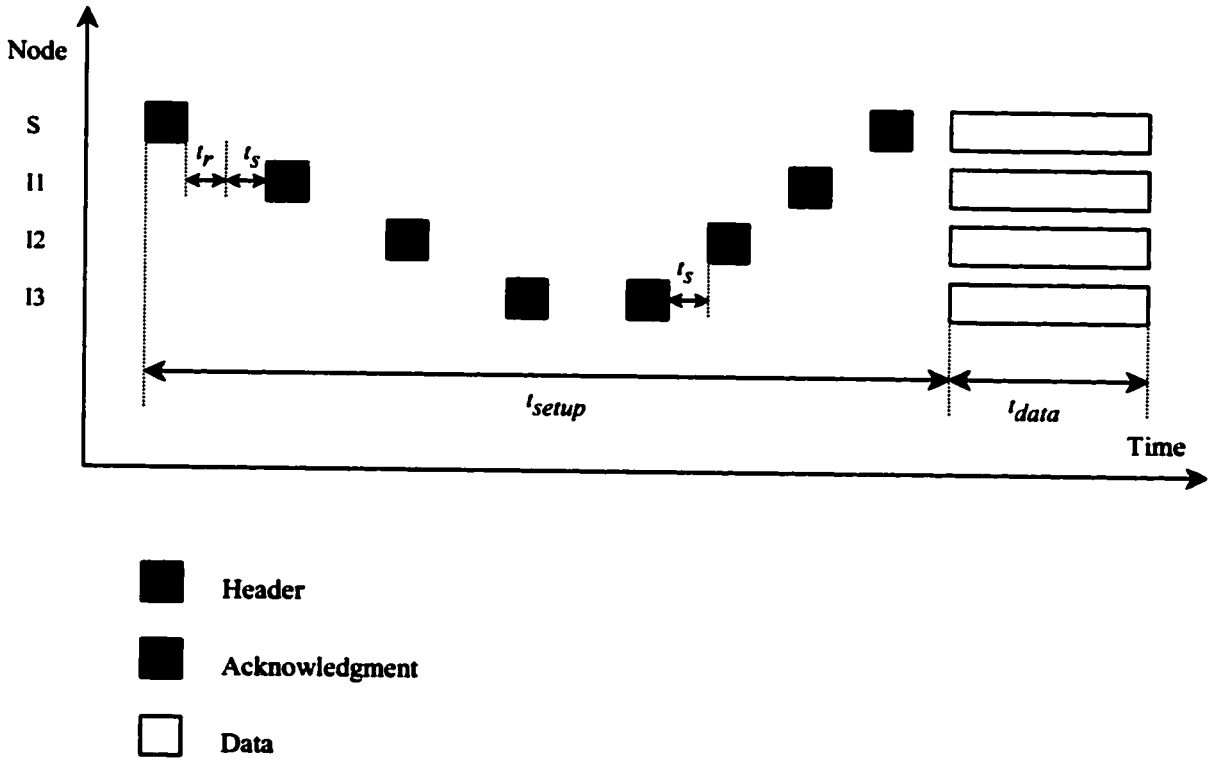


Figure 2.2 Circuit switching

The no-load latency, as can be seen in the figure, can be computed by combining the path setup time, and the data transmission time:

$$t_{circuit} = t_{setup} + t_{data}$$

$$t_{setup} = D[t_r + 2(t_s + t_w)]$$

$$t_{data} = \frac{1}{F}\left\lceil \frac{L}{W} \right\rceil$$

Once the channel is reserved it has the advantage of guaranteeing a certain amount of bandwidth and low latency communication between the two nodes. But this tends to waste more resources than is necessary, as that channel medium might be reserved without having any messages passing through it in certain instances. Also if the header becomes blocked waiting on some resources, then the other packets in the network cannot utilize all the channels already reserved by it. Therefore circuit switching is more appropriate when the messages are long and infrequent, so that the path setup time is hidden by the data transmission time [4, 14].

2.3.2 Packet switching

Packet switching networks do not require dedicated channels for communication between nodes. Rather, the message is partitioned into fixed size packets, with the routing information contained in the first few bytes of the packet that is referred to as the header. Source nodes initiate communication by sending the packet to a neighboring node along the path to the destination. Each intermediate node must receive the whole packet before it can send it along its way to the next node, until the packet reaches its destination. This is why this switching technique is also referred to as *store-and-forward* switching. Packet switching requires that each node be capable of buffering whole packets, which increases the buffering requirements of the nodes and respectively their hardware complexity. This accordingly leads to slower network speeds. Initial implementations of packet switching, and in a carry over from local area networks, buffered the packets in the local memory of the node. As performance became an issue, the delay of storing and retrieving the packets from local memory was not acceptable, so packets were buffered and managed directly within the router. Figure 2.3 below provides a time diagram for the operation of this switching technique.

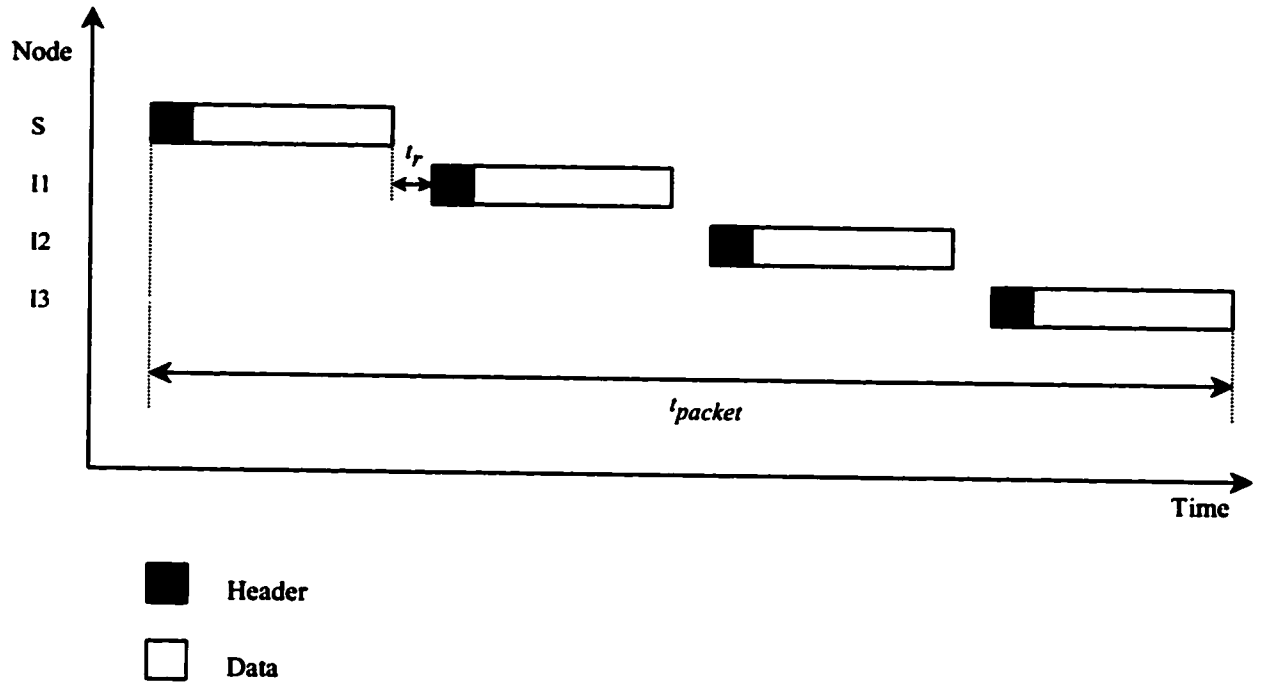


Figure 2.3 Packet switching

The no-load latency for packet switching can be derived using the following formula:

$$t_{packet} = D\left\{t_r + (t_s + t_w)\left\lceil \frac{L+W}{W} \right\rceil\right\}$$

Packet switching overcomes the problem of reserving channels without utilizing them, as was the case for circuit switching. Packets can utilize any available channel along their paths. But the requirements to buffer entire multiple packets simultaneously can be quite expensive. This switching technique therefore is more suitable for networks with packets that are frequent and relatively short. Also the latency in these networks grows with the diameter of the network due to the repetitive process

of storing the whole packet in each node along the way [4, 14].

2.3.3 Virtual-cut-through Switching

This technique was developed to overcome the latency problem of packet-switched networks. In this technique an intermediate node along the path attempts to forward the packet along its way to the next node as soon as it receives only the necessary information needed to route that packet, which is mainly the header that contains the address of the destination node. By not having to wait for the remainder of the packet to be received, the latency is reduced. A time diagram illustrating the operation of this switching technique is provided below in Figure 2.4. In the diagram it can be seen that the packet is blocked before reaching intermediate router *I*1 along its path due to a busy channel. But the packet manages to cut through in successive intermediate routers.

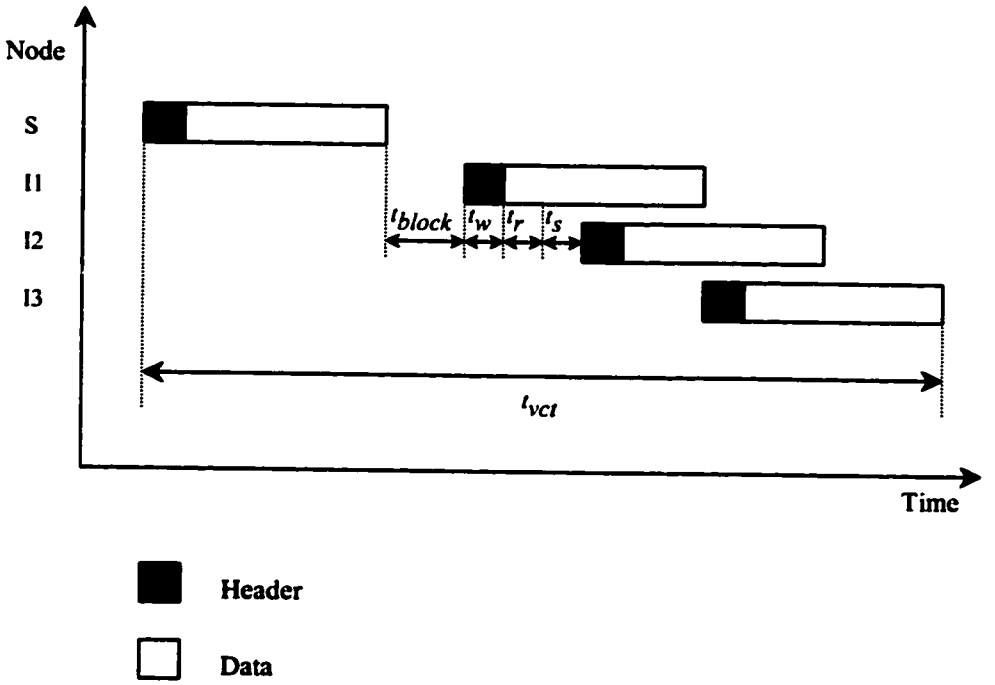


Figure 2.4 Virtual-cut-through switching

The no-load latency can then be derived using:

$$t_{vct} = D(t_r + t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{L}{W} \right\rceil$$

In virtual-cut-through switching, whenever the header of the packet is blocked at a certain node, the tail of the packet continues to advance towards that node. If the header is blocked long enough, the whole packet will be buffered at that node. This means that the buffering requirements of virtual-cut-through networks are still the same as those in packet-switched networks [4, 19].

2.3.4 Wormhole Switching

Wormhole switched networks use the virtual-cut-through concept. Each packet is divided into smaller *f*low control dig*its* (*flits*). The header flit(s) governs the route of the packet. An intermediate node attempts to forward the packet to the next node as soon as it receives the header flit. Once the header flit is routed through a particular channel, the remaining flits must follow in a pipeline fashion. Flit interleaving is not allowed, as they do not contain any routing information. The main difference between wormhole and virtual-cut-through becomes apparent when the header of a packet is blocked. In this case, wormhole routing uses flow control to block the progress of the packet and freeze it throughout the network. This feature is what gives this technique its name, as the packet resembles a worm stretched out in the network. A time diagram for the operation of wormhole switching is shown in Figure 2.5 below.

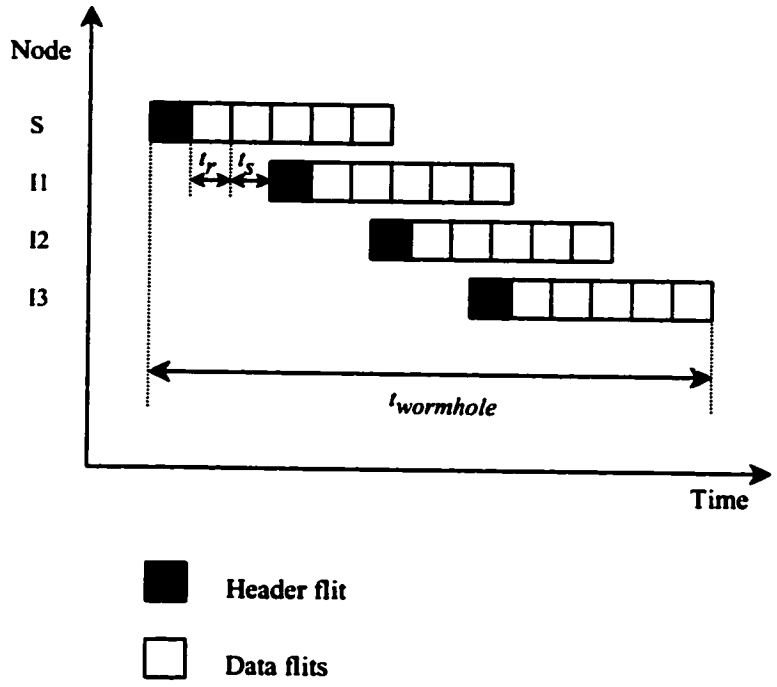


Figure 2.5 Wormhole switching

The base latency for wormhole switching can be derived using:

$$t_{wormhole} = D(t_r + t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{L}{W} \right\rceil$$

The latency expression is identical to that of the virtual-cut-through switching as no contention is taken into account. A major distinction between the two switching schemes however, is that the unit of flow control in virtual-cut-through is the whole packet, while it is based on a single flit in wormhole switched networks.

The main advantage of wormhole switching is reduced buffer requirements to, as low as, one flit buffer per channel within each node. This enables the design and construction of nodes that have much simpler hardware, low cost, and can run at higher clock speeds. Add to that the low latency communication that does not grow with higher diameter networks, it becomes clear why wormhole networks have received a great deal of attention and subjected to a good deal of width and breadth of research in this field of study. Most commercial parallel multiprocessor and multicomputer machines on the market today use wormhole switching in one form or another. Its success is evident in expanding its use to Distributed Shared Memory (DSM) architectures, ccNUMA architectures, NAS (Network Attached Storage) devices, and LAN areas, such as the Network of Workstations (NOWs), Myrinet, and the ServerNet projects [5, 9, 12, 14, 20, 21, 22].

2.3.5 Hybrid switching

Hybrid switching is accomplished by dynamically combining wormhole switching and another complementary switching technique on the same router architecture. By combining the advantages of two different switching techniques, or using one switching technique in circumstances where the other shows undesirable characteristics, a new hybrid switching technique is obtained. The new switching technique demonstrates favorable operational characteristics than either of the switching techniques implemented separately.

Wormhole switching generally suffers from high channel contention at high traffic rates, as blocked packets maintain their channel resources. This leads to lower achievable throughputs of wormhole switched networks. On the other hand, it can achieve very fast switching of flits due to its light and efficient buffer requirements. In contrast, packet and virtual-cut-through switching can achieve higher throughputs at the expense of allocating large and more expensive memory buffers that must store whole packets at each intermediate node. Circuit switching attempts to provide a dedicated channel of communication between nodes, at the expense of longer path setup times and possible inefficient use of

channel resources if dedicated channels are reserved but not used.

Hybrid switching techniques consequently attempt to improve either the low throughput problem of wormhole switching by providing some buffering mechanism for blocked packets in order to alleviate and better utilize the channel bandwidth available to the network. Or they try to improve the reliability and fault tolerance features of the network. These techniques generally attempt to find the right balance between using channel resources and memory resources in order to provide higher throughput performance at reasonable hardware cost.

2.3.5.1 Buffered Wormhole Switching

Buffered wormhole switching technique was proposed by IBM and is utilized in their successful Power Parallel SP systems. This hybrid switching technique combines features of wormhole and virtual-cut-through switching on the same switching elements. The interconnection network is a multistage generalized Omega network using 4x4 crossbar switches with bidirectional links. A frame is constructed using a two-staged MIN; each stage has 4 crossbars for a total of 8. Each frame can therefore support 16 nodes. Routing in SP systems is source-based, with each routing flit containing the output ports to be utilized along successive intermediate nodes. The first flit in the packet contains the packet length, with routing flits immediately following it. One routing flit is required per frame, therefore larger systems constructed with more than one frame, require one routing flit per each additional frame.

Buffered wormhole switching allocates a local central memory that is shared by all input and output ports of the switch. This central memory is dynamically allocated and organized as a linked list. Each list is associated with an output queue.

When routed packets request output channels that are available, they are advanced using the basic wormhole switching mechanism. When a packet is blocked, a portion of that packet is buffered at the local memory allocated for that purpose. When blockage occurs, the flits of a packet are grouped in a *chunk* at the input queue, with each chunk containing 8 flits. This chunk is then transferred to the local

memory in a single clock cycle. The central memory can support 218 chunks. One chunk is allocated for each output queue initially, while the remaining chunks are allocated to blocked packets dynamically and upon demand. When the central memory fills up, the switch operation is restricted to wormhole switching.

In the case where the packets are short or enough memory is available, the whole packet may be buffered when blocked, which resembles the operation of packet and virtual-cut-through switching. Buffered wormhole switching is distinguished from virtual-cut-through as the basic flow control is still implemented at the flit level for normal operation, and at the chunk level for transferring flits in and out of the central memory [5, 23].

2.3.5.2 Pipelined Circuit Switching

Pipelined circuit switching combines features of wormhole switching and circuit switching in order to provide a switching mechanism that is more reliable and fault tolerant against failing routers and channel components. This switching mechanism is distinguished from circuit switching as the reserved path here is established on a virtual channel, rather than on a physical channel level.

In wormhole switching, if the header of a packet is blocked due to a faulty channel, then the packet may block indefinitely in the network. Adaptive and nonminimal adaptive algorithms provide some flexibility in avoiding faulty resources, but they do not guarantee the delivery of packets in the presence of faults in all cases.

Pipelined circuit switching attempts to establish and reserve a safe path between source and destination nodes before the data is injected into the network. First the header flit is sent to find a reliable path. If the header is blocked due to a faulty channel, the header can freely backtrack to the previous node and selects another available output channel in an attempt to avoid the faulty component. When the header finally reaches its destination, an *acknowledgment flit* is sent back to the source node. At that point the source sends the data flits over the reserved path between source and destination nodes

in a wormhole pipelined fashion. This switching mechanism also requires extra virtual channels, as data channels are separated from control virtual channels, which are used to transmit the header and acknowledgment flits. The switching mechanism requires two additional virtual channels for each data channel, for a total of three virtual channels per virtual channel used in normal wormhole switching. One virtual channel is used for advancing headers, the second is used by the acknowledgment flits and backtracking headers. The main virtual channel is used for the transfer of the data flits. If the control virtual channels are multiplexed on the same virtual channel, then this mechanism requires one additional virtual channel in each direction between adjacent nodes. One implementation, the Ariadne router utilizes two data channels, and one control virtual channel for each physical channel. As in the case of circuit switching where headers do not block, freedom of deadlock is achieved without implementing any routing restrictions.

The reliability achieved by pipelined circuit switching is not without cost, namely larger latencies are incurred due to the path setup delays in comparison with wormhole switched networks. There are variations of this switching mechanism that attempt to reduce the impact of path setup delays on the performance of the network. *Scouting Switching*, for example, allows the data flits to be transmitted following the header flit, as long as they maintain a certain distance, called the *scouting distance*, away from the header to allow for backtracking maneuvers by the header if necessary. The scouting distance can be modified in order to achieve the required balance between fault tolerance and performance. If the scouting distance is set to zero, then the switching mechanism reverts back to basic wormhole switching [4].

2.3.5.3 Wave Switching

Wave switching proposes using circuit switching and wormhole switching concurrently on the same network topology. The main objective of wave switching is to more effectively take advantage of temporal and spatial communication locality amongst nodes within the message-passing network.

Circuits are pre-established between nodes that intend to communicate frequently, while the other nodes communicate using wormhole switching. This technique requires a new router architecture that can support both switching techniques simultaneously. Various switches and routing units are therefore used to implement both switching techniques. The switches that implement circuit switching adopt the pipelined circuit switching technique discussed earlier, except that the paths here are established over physical channels that are made narrower as data channels are separated from control channels.

In order to achieve very high channel utilization and lower communication latencies, channel pipelining is adopted for the established path circuits. Using this technique new data are injected into the channel before the previously injected data have cleared the channel. Propagation speed becomes only limited by the wire capacitance. More so, the flit buffers are not needed on the physical channels used for circuit switching, and are therefore removed. Consequently the flow control mechanism is also removed, but synchronizers are then required at each delivery channel in order to manage skew on the parallel wires. This even makes higher clock frequencies that can be used to drive data across these channels more feasible. Channel pipelining is implemented across both channels and switches, and therefore the name wave pipelining or wave switching. These new switching techniques also use their own routing algorithms and protocols such *as Cache-like Routing Protocol* (*CLRP*), and *Compiler Aided Routing Protocol* (*CARP*) [24, 25].

There has been various other hybrid techniques proposed in the literature. One implementation proposed in [26] combines features of wormhole switching and virtual-cut-through switching in order to provide higher achievable throughput, while still reducing the buffer storage requirements by virtual-cut-through routers. The technique attempts to relieve contention at high loads by selectively buffering those packets that are blocked while occupying a high number of links in the network. A new field is added to the header that counts the number of hops traversed by the packet. If the number if hops exceeds a predetermined threshold, then the packet is buffered when it is blocked. A multi-purpose network adapter was developed and utilized in this study. SPIDER (*Scalable Point-to-point Interface*

DrivER) can support multiple switching techniques, such as packet switching, virtual-cut-through, and wormhole switching [26, 27].

Although hybrid techniques generally achieve higher performance when wormhole switching is strictly adopted. There is still a lot of room for improvement within the wormhole switching field that will result in improving the performance of wormhole switching whether used separately or in conjunction with other switching mechanisms.

2.4 Flow Control

Flow control manages the allocation of the required resources by a packet while it is being transmitted through the network and until it reaches its destination. Communication between nodes is performed by sending messages, which may be broken down into one or more packets for transmission. A packet is further divided into flits. Flow control is performed at the flit level, while routing is performed at the packet level. Information is sent over the physical channels in *phits* (*ph*ysical transfer un*its*), which are usually made to be of the same size or smaller than flits. A phit can be transferred across the physical channel in one clock cycle. Therefore, if a flit is broken down into smaller phits depending on physical channel width limitations, multiple cycles may be needed in order to transfer a single flit across the physical channel.

The flow control protocol synchronizes the transmission and reception of flits between adjacent nodes. It determines how buffers and channel bandwidth are allocated, and how packet collisions over resources are resolved. A resource collision occurs when a packet P is unable to progress because it requires a buffer resource that is occupied by another packet. Flow control can resolve collisions by adopting one of the following policies:

1) Block packet P.

2) Buffer packet P in the previous node.

3) Drop packet P.

4) Misroute packet *P* through any available channel. The channel may not be profitable for *P*.

Depending on which of the above choices are implemented, an input selection policy might be also implemented by the flow control scheme in order to determine which packets may use the output channel, such as FCFS (First Come First Serve), round robin, and fixed channel priority.

Backpressure is the most commonly used flow control mechanism for wormhole switched networks. The main idea is that when a node runs short on input buffers it sends a flow control signal to the previous node to stop transmission of flits across the physical channel and block the packet in place at that node. The timing of when to send that flow control signal must take into account the propagation delay of the physical channel. This means that the node must insure the availability of enough free buffers in order to store any flits that were sent or are in transit before the flow control signal was received by the previous node. The flow control mechanism can be implemented in an asynchronous or synchronous fashion. A simple four-phase asynchronous request/acknowledge handshaking protocol is sufficient to implement this flow control mechanism [4, 9].

When the flow control strategy allocates buffers and channel bandwidth to flits, it needs a mechanism to associate between those flits and a particular packet. This is because flits contain no routing or sequencing information. Flow control strategies are tightly coupled to the operation of wormhole switching, in contrast to buffered packet switching that do not depend on any one particular flow control mechanism. Also the importance of congestion control is reduced in comparison with buffered networks, as the flit-level flow control quickly gives a backpressure signal to source nodes in order to reduce traffic.

When virtual channels are implemented, the virtual channel is used to associate a packet with a set of buffers along with some control state. A virtual channel is allocated to a packet, while the buffers of the virtual channel are allocated in a FIFO manner to the flits of that packet. More generally when virtual channels are used, a virtual channel flow control is needed in order to assign virtual channels at the packet level, and allocate physical channel bandwidth at the flit level.

The main idea behind wormhole switching is to relieve the switching devices from buffer

management, which facilitates the VLSI design of compact and fast switching devices. This does not come for free, and the price to pay is in the implementation cost of the flit-level flow control. Generally a good flow control policy should avoid channel congestion while reducing the network latency [14, 18, 20, 28].

2.5 Routing

In the absence of a complete topology, where every node has a direct connection to every other node within the network, routing determines the path selected for each packet to reach its destination. Efficient routing is critical to the operation of direct networks. The routing mechanism cannot be considered in exclusion of the topology of the network. Various topologies may either tolerate certain characteristics of the routing scheme, or dictate others.

Routing can be decomposed into two separate functions, the routing function, and the selection function. The *routing function* determines a set of possible output channels for routing a packet at each intermediate node along its path. The routing function takes into account the current and destination nodes of the packet. It is responsible for implementing the logic of the routing algorithm in use. The *selection function* takes the set of possible output channels produced by the routing function and selects a single channel from that set to route the packet through. The decision made by the selection function is determined by the status of output channels at a particular intermediate node. The selection function attempts to select a free output channel if possible from this set. If more than one output channel is available, then the function chooses amongst them according to a certain flavor of selection. There are many flavors of selection that have been proposed in the literature, such as the *straight-first*, *turn-first*, and *random* selection functions. The straight-first selection function favors those channels that cause the packet to remain in the same dimension it was traveling in. The opposite turn-first function favors those channels that cause the packet to turn or change dimensions at that point, while the random function selects amongst the channels in a random fashion. The selection function is invoked only when

there are two or more possible output channels. The selection function does not interfere with the properties of the routing algorithm such as freedom of deadlock, but it does affect the overall performance of the network. No single selection function has been found to be suitable for all kinds of network topologies and traffic patterns, more so tuning the selection function to the applied workload can significantly improve its performance [29].

Routing functions can either be *source-based* or *distributed*. In source-based schemes, the source node specifies the entire routing path. It is analogous to street-sign directions that a packet will follow until it reaches its destination. Source-based routing incurs more overhead due to larger packet header sizes for longer routes, and is therefore not suitable for large diameter networks.

In distributed routing, each intermediate node along the path determines where the packet will be forwarded to next. In this case only the destination node address needs to be stored in the packet header. Each intermediate node should make the routing decision based only on local information for obvious performance reasons [5, 14].

Table lookup routing has been utilized lately by routing schemes in order to deal with irregular network topologies present in LANs, such as Myrinet, Autonet, ATOMIC, and ServerNet [10, 11, 12]. Table lookup can be used by either the source-based or distributed routing schemes, but it is mainly associated with the latter [13]. Regular network topologies do not normally make use of table lookups unless a more fault resilient routing scheme is desired. In distributed routing schemes with table lookup, each intermediate router consults its routing table to determine the packet destination. The routing tables must be created and maintained by a distributed routing algorithm. Table lookup routing provides irregular networks with a low packet overhead mechanism by using only the destination node address to index the routing tables. The fixed size of such tables puts a limit on the number of nodes in the network, and consequently makes this scheme ineffective for large size or MPP interconnection networks [5, 15].

2.6 Definitions and Concepts

This section provides a definition of various problems that must be protected against, and a major concept used within wormhole routed networks, namely virtual channels.

There are various potential problems that can be encountered while routing packets. For satisfactory and sane operation of the network, the routing algorithm must address and resolve these problems. These problems are *deadlock*, *livelock*, and *starvation*. Definition of these terms for this context is provided next.

2.6.1 Deadlock

Deadlock occurs when a set of packets becomes blocked forever in the network. Deadlock can potentially take place whenever packets do not release previously reserved network resources when they are blocked. Wormhole routing is particularly susceptible to deadlock formations because blocked packets are not removed from the network, and hence the channels and flit buffers they occupy are not released. Deadlock is alternatively defined as waiting for an event that cannot happen. An example is illustrated in Figure 2.6.

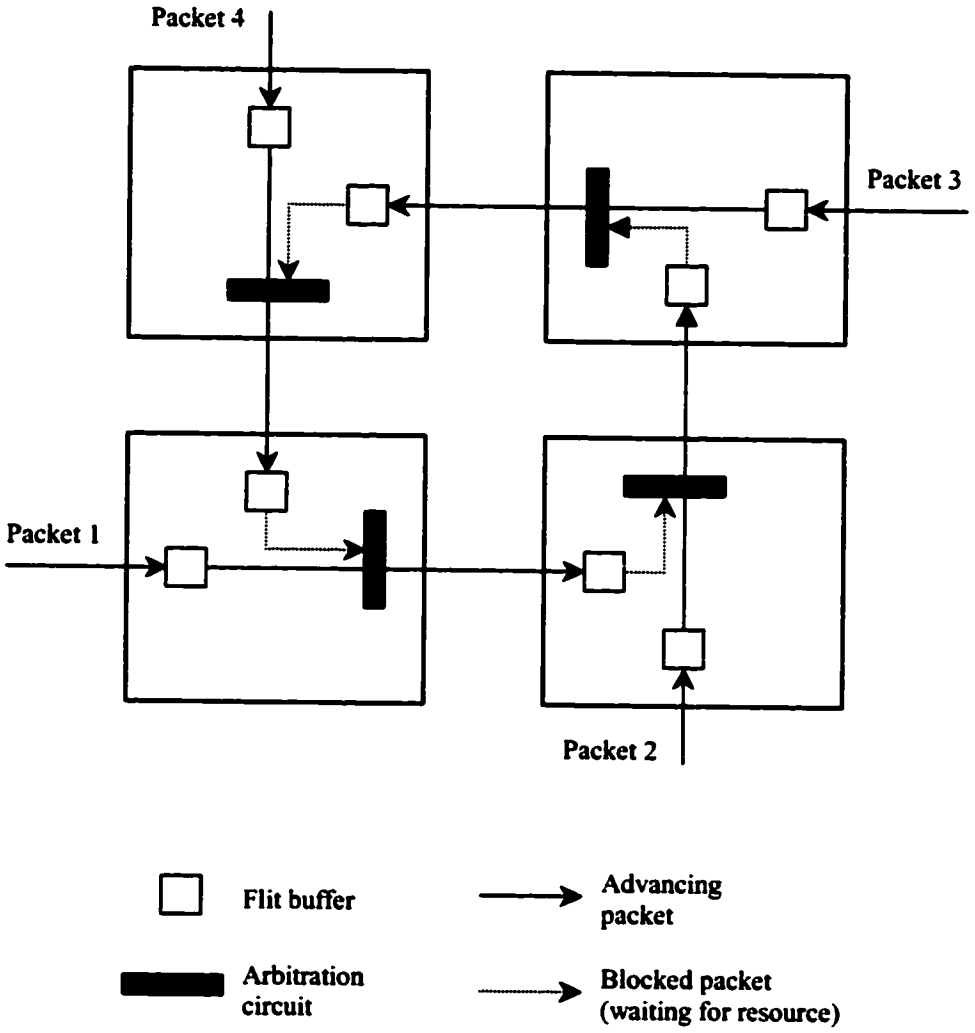


Figure 2.6. Example of a deadlock cycle involving four packets.

Deadlock cycles can also be classified into single-cycle and multi-cycle deadlocks depending on what is referred to as knot cycle density. Knot cycle density represents the number of unique cycles that form the deadlock. Single-cycle deadlocks can be formed mostly in networks that do not implement virtual channels (explained in section 2.6.4 below). They can also exist with a much lower degree of

probability, and in rare conditions, in networks that incorporate multiple virtual channels and that allow full routing freedom on those virtual channels. Multi-cycle deadlocks on the other hand can only occur in networks that support virtual channels, as two or more cycles that form the deadlock do in fact exist on two or more separate sets of virtual channels [30, 31]. While Figure 2.6 demonstrates a generic single-cycle deadlock configuration, Figure 2.8 that will be provided in section 2.6.4 will illustrate a multi-cycle deadlock configuration.

Preemption of packets involved in a deadlock situation can provide a solution to the deadlock problem. Customarily, packets involved in a deadlock are either misrouted or dropped altogether. Allowing the packets to be misrouted is basically what nonminimal routing algorithms implement. This type of routing will be discussed in the next chapter. Discarding the packets requires a more elaborate scheme of relaying what has transpired to the source node so that it can reschedule them for retransmission. This alternative can lead to high network latency and reliability problems, and is therefore not usually implemented [14]. Alternative solution adopted to tackle this problem will be considered in more detail in the next chapter.

2.6.2 Livelock

Livelock occurs when the routing of a packet never leads it to its destination. In other words the progress of a packet does not cease, nonetheless the packet does not reach its destination. Livelock is possible only when the routing technique is adaptive and nonminimal, or when misrouting is allowed without any restrictions [32].

2.6.3 Starvation

Similar to deadlock but occurs when a packet waits for an event that can happen but never does. In

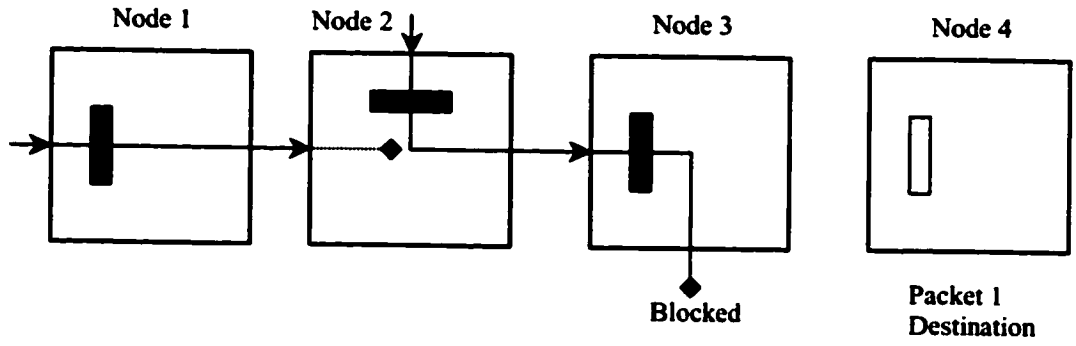
other words, a packet may wait forever trying to acquire a resource, but other packets are always successful in grabbing it first [32].

2.6.4 Virtual Channels

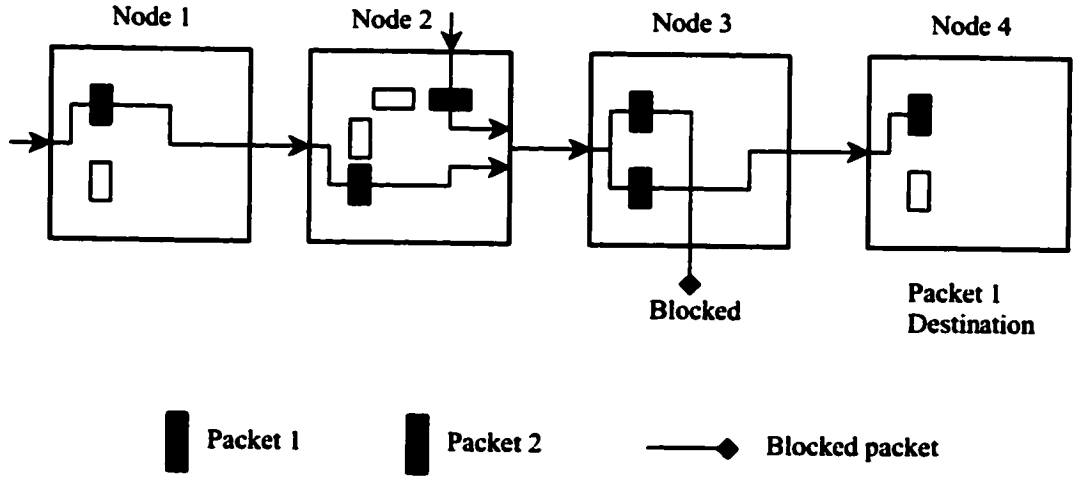
Virtual channels were initially proposed by Dally and Seitz in [33] in order to design deadlock-free routing algorithms. The most expensive resources in an interconnection network are the wires that provide the physical channel bandwidth. The second most expensive resource is buffer memory. Adding virtual channels to a network makes more effective use of both of these resources by decoupling their allocation at the minor expense of a small amount of added control logic. More specifically virtual channels decouple the allocation of buffers from the allocation of channels by providing multiple buffers for each physical channel in the network. Each virtual channel has its own flit buffers, control, and data paths. Several virtual channels are multiplexed on a single physical channel.

Dally in [33] provides the following analogy: Adding virtual channels to an interconnection network is analogous to adding lanes to a street. A network without virtual channels is composed of one-lane streets. In such a network, a single blocked packet blocks all the trailing packets. Adding virtual channels to the network adds lanes to the streets allowing blocked packets to be bypassed.
Dally has shown in [28] that the use of virtual channels can increase the throughput of various interconnection networks from 50% to 90%. Figure 2.7 provides an illustration of the operation of virtual channels. In Figure 2.7a, packet 1 is blocked awaiting the buffer resource, which is reserved by packet 2. In this case packet 1 remains blocked and waiting as long as packet 2 remains blocked. This situation does not lead to the effective utilization of the channel and buffer resources. When virtual channels are used as in Figure 2.7b, packet 1 can now bypass blocked packet 2 at node 2. This is due to the fact that a virtual channel in node 3 is free and can receive flits from packet 1. At node 2, virtual channels are multiplexed on the same physical channel. In other words the physical channel bandwidth is allocated to one of the virtual channel buffers at a time, assuming that the buffer has some data to send. If more than one virtual channel buffer has data to send, the physical channel bandwidth is

alternated amongst all the occupied buffers that are requesting the same physical channel. In this network with virtual channels, packet 1 is able to advance and reach its destination, while packet 2 remains blocked. This tends to reduce the contention and increase the performance of the network.



(a) Packet 1 is blocked behind packet 2 while all physical channels remain idle.



(b) Packet 1 bypasses the blocked packet 2 and advances to its destination.

Figure 2.7. Virtual channel operation.

The term virtual channels is adopted when these channels are used for the purpose of providing deadlock free routing algorithms, otherwise, if they are used solely to provide extra adaptivity, they are referred to as virtual lanes. Virtual channels are also sometimes referred to as edge buffers.

In addition to increasing throughput, virtual channels provide an additional degree of freedom in allocating resources to packets in the network. Also virtual channels allow a physical network to be partitioned into multiple disjoint logical networks, which is required by most adaptive routing algorithms. Virtual channels increase the amount of flexibility in a network in various ways. First, applications can be designed using any logical topology. This logical topology can then be mapped onto the physical topology of any physical network by adding virtual channels wherever required. Second, adaptive routing requires extra virtual channels for congestion control, and fault tolerance purposes. Finally, for certain application, virtual channels assist in allocating the minimum required level of communication bandwidth.

The virtual channel concept has some drawbacks. A trade-off between increased network throughput and longer communication latency should be considered. For example adding virtual lanes was found to be more effective in increasing throughput than increasing the depth of the buffers [28]. But as the number of virtual channels increase, the scheduling becomes more complicated requiring additional hardware complexity and potentially increasing network latency. In [34] it was shown that routers based on virtual channels require 2 to 3 times as many gates as the routers that are not based on virtual channels. Routers with virtual channels also have setup delays 1 to 2 times higher, and have flow control cycles 1.5 to 2 times longer. The study also showed that each added virtual channel should provide 30-50% increase in physical channel utilization in order to offset the accompanying expected decrease in network clock speed [28, 34].

Now that the concept of virtual channels is explained, a multi-cycle deadlock is illustrated in Figure 2.8 below. Multi-cycle deadlocks can only occur in networks with virtual channel support.

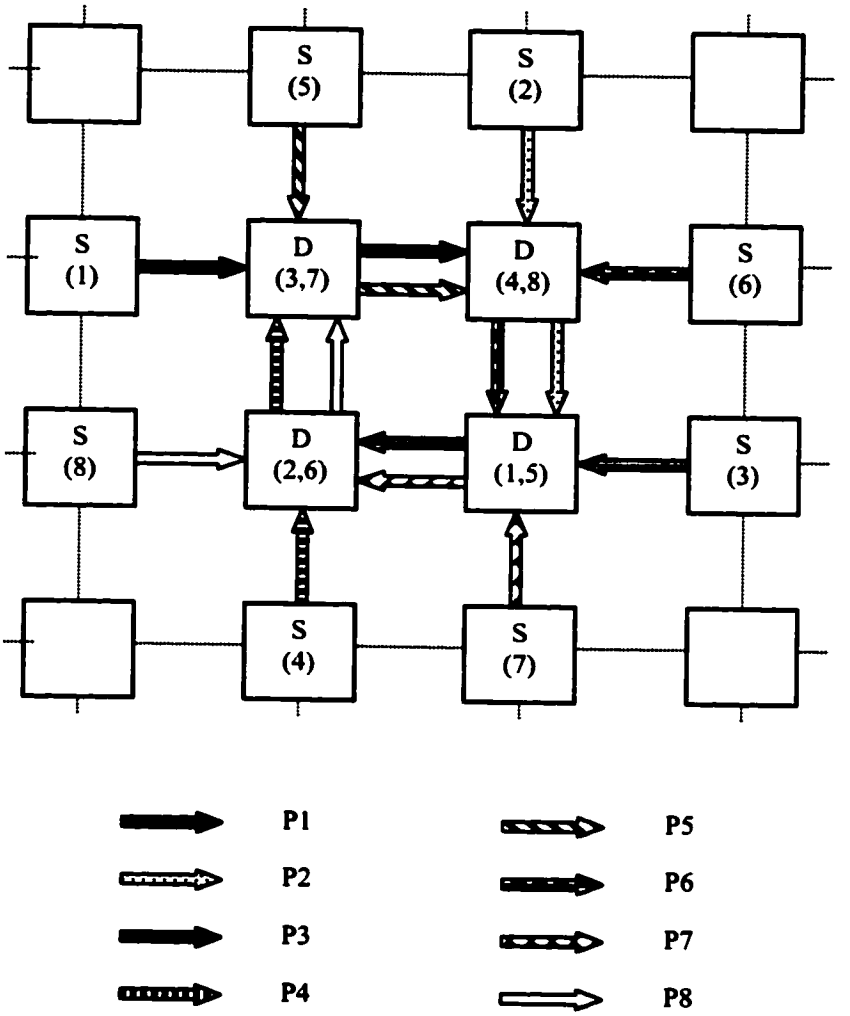


Figure 2.8. Multi-cycle deadlock configuration with two virtual channels

CHAPTER 3

RELATED WORK: ROUTING ALGORITHMS

Routing algorithms used to route packets in wormhole networks are broadly classified into two major categories, *deterministic* and *adaptive* routing algorithms. These algorithms are further characterized by the amount of adaptivity they provide, and by the way they protect against deadlock. Definitions of these terms are presented next, followed by some routing algorithm examples.

1.1 Deterministic Algorithms

Deterministic algorithms do not provide any adaptivity, and are therefore *oblivious* to the state of the network. The route taken by a packet is determined by considering only the source and destination node addresses. Deterministic routers allow for the design of fast and simple routers, as each intermediate node performs very simple if any routing decisions. In the case of *source routing*, the source node constructs the entire path that the packet must follow along its route. Intermediate nodes merely follow instructions on where to route the packet next. While in *distributed routing*, each intermediate node along the path executes very simple routing decisions. Due to its simplicity, the routing algorithm can be implemented in fast and compact hardware such as finite-state machines, or routing logic. As a consequence, most first generation commercial and development multicomputers used deterministic routing algorithms, such as the Intel Paragon, Cray T3D, nCUBE-2/3, Stanford DASH, and the MIT J-Machine [4, 14, 33]. Deterministic routers suffer a serious drawback however; they do not allow the router to react to the status of the network. This means that it cannot route around congested or faulty nodes

3.2 Adaptive Algorithms

Adaptive routing algorithms on the other hand can consider and react to network conditions by definition as they allow a packet to be routed along alternate paths. The routing decision is performed in a *distributed routing* fashion by each intermediate node along the way. Several development and commercial parallel machines use adaptive routing, such as the Reliable Router, the S-Connect system, and the Cray T3E [4, 35]. Next generation multicomputers are more likely to implement adaptive routing as well. Adaptive algorithms are further classified as to the amount and type of adaptivity they provide.

3.2.1 Fully Adaptive Routing Algorithms

Fully adaptive routing algorithms allow a packet to be routed on all possible channels throughout its path to the destination. Possible channels for a packet exclude those channel resources that are dedicated for deadlock avoidance. This is why these routing algorithms are more appropriately referred to as *maximally adaptive* algorithms.

Fully adaptive routers provide the most routing flexibility for the class of deadlock-avoidance algorithms. Higher routing flexibility increases the probability of deadlock formations proportionally and hence there is an associated increase in hardware requirements for deadlock prevention. The cost of resources dedicated to deadlock avoidance compared to those used to perform the actual adaptive routing produces expensive and inefficient router designs, which makes this type of routing generally impractical for multicomputers implementations [4, 36].

3.2.2 Partially Adaptive Routing Algorithms

Algorithms that provide limited adaptivity are called *partially adaptive* algorithms. As mentioned previously, although fully adaptive routers have the highest routing flexibility, they require complex and costly router designs. This has triggered various attempts to curtail the amount of adaptivity provided by the routing algorithm, in an effort to lower resource requirements, and therefore to a simpler, more affordable router designs [14].

3.2.3 True Fully Adaptive Routing Algorithms

To overcome the high resource requirements of avoiding deadlocks in fully adaptive routers, and achieve even higher routing flexibility, deadlock recovery mechanisms are used along with fully adaptive routers. In this case no channel resources are reserved for deadlock avoidance. All possible paths are fully dedicated for routing packets, and hence these algorithms are called *True fully Adaptive Routing* (*TFAR*) Algorithms [4].

3.2.4 Minimal Routing

Minimal routing algorithms are only allowed to route packets along available shortest paths between any pair of nodes. This implies that each routing step taken by the algorithm must bring the packet closer to its destination, which is why these algorithms are also called *profitable* routing algorithms.

Deterministic routing algorithms are usually minimal. Whenever fully adaptive routing algorithms use a minimal strategy, they are called *fully adaptive minimal* routing algorithms [4].

3.2.5 Nonminimal Routing

Nonminimal routing or *misrouting* may present a packet with path selections that are not profitable, or may direct the packet farther away from its destination. Misrouting is mainly used to provide fault tolerance, and to route around heavily congested areas of the network. The packet in this case may wander forever, which means that misrouting must protect against the possibility of livelock as well as deadlock [4].

3.3 Deadlock Resolution

Deadlock presents the most difficult problem to be resolved in wormhole routed networks. As mentioned earlier, deadlocks occur in wormhole routing because packets acquire additional resources as they advance, without necessarily releasing what they currently hold. When two or more packets are waiting on resources that are currently reserved by each other, the packets block forever in a deadlock formation. Deadlock in virtual-cut-through switching can be dealt with much more easily than wormhole switching, as packets do not reserve their channel resources when they are blocked. Algorithms in virtual-cut-through networks can use the *deflection routing* (also known as *hot potato*) mechanism in order to avoid deadlock by continuously misrouting a packet whenever the channel resource required by that packet is occupied at a particular node [4].

Wormhole routing algorithms mainly protect against deadlock in two fundamental ways. They either employ *deadlock-avoidance* or *deadlock-recovery* techniques [4]. These two techniques are similar in the sense that in each case the routing algorithm must dedicate or resort to some special resources that are used for the avoidance, or recovery from deadlocks respectively. But they differ in the amount, expense, and impact of these resources on the total complexity and speed of the router.

3.3.1 Deadlock-avoidance Algorithms

The basic idea in this technique is to prevent deadlocks from occurring altogether. This is accomplished by devising and enforcing a set of restrictions on the routing algorithm, such that all the resources presented to an advancing packet cannot lead to a deadlock situation. Various theoretical models have been proposed in the literature in order to assist in the development of deadlock-free routing algorithms. Originally, the main concern of such models was to provide a sufficient condition that routing algorithms should uphold in order to achieve deadlock freedom, regardless of the amount and expense of the resource requirements. Later models were refined in order to allow maximal adaptivity using a minimal number of channel resources. These models provide a necessary and sufficient condition for achieving freedom from deadlock. All these models serve as a framework for developing various routing algorithms, and are reviewed in more detail next [33, 37, 38].

3.3.1.1 Channel Dependency Graph Model

This model uses the concept of *channel dependency graphs* and also makes use of virtual channels. Developing deadlock-free routing algorithms using channel dependency graphs was shown in [33]. The channel dependency graph for a direct network, and a particular routing algorithm is defined as a directed graph $D = G(C, E)$, where the vertices of D consist of all the unidirectional channels in the interconnection network I. The edges of D are all the pairs of channels (c_i, c_j) such that there is a channel dependency from c_i to c_j. Using this technique, the physical channels of the channel dependency graph that form cycles are split into different groups of virtual channels. The virtual channels in the network are then structured so that the routing algorithm allocates them to advancing packets in a decreasing order so as to eliminate any cycles in the channel dependency graph. Extending the network with additional virtual channels is necessary only to make the resulting routing function

connected, and in some rare cases they may not be needed.

The prevention of cyclic dependencies, a precondition to deadlock, from forming amongst the packets avoids the occurrence of deadlocks. It was proved in [33] that a routing algorithm is deadlock-free if there are no cycles in its channel dependency graph *D*.

Many deterministic and some adaptive routing algorithms were developed using this model. The concept of virtual channel groups used here was expanded to virtual networks and was used to develop fully adaptive routing algorithms. Because this model lends itself toward the development of deterministic routing algorithms, the extension of the model to develop adaptive and fully adaptive routing algorithms produces somewhat inefficient and sometimes very expensive routing functions [14, 17, 32, 33].

3.3.1.2 Extended Channel Dependency Graph Model

This model is usually referred to as the *Duato Protocol*. It was developed by Duato in [37, 39]. The work was motivated by the observation that although the absence of cyclic dependencies is a necessary and sufficient condition for deadlock-free deterministic routing, it is only a sufficient condition for deadlock-free adaptive routing. Adaptive routing has been shown to outperform static algorithms. Also in [34] it was shown that extra virtual channels can increase the hardware complexity considerably with adverse effects on performance. Consequently, the need for highly adaptive algorithms that require a minimal number of virtual channels becomes a goal worth achieving and a driving force behind developing this theory. In deterministic routing, each packet has only one possible routing option at each node, and therefore the algorithm requires that all cyclic dependencies amongst channels be eliminated in order to guarantee freedom of deadlock. Adaptive routing on the other hand provides more routing choices to each packet at each node, and therefore it does not become necessary to eliminate all cyclic dependencies. This remains true so long as there exists a subset of the channels that do not have any cyclic dependencies and that can be used by packets to eventually reach their

destinations. This subset of channels is provided by a routing subfunction and is called the escape channels. When a packet is blocked it uses the escape channels, it can however return to use the remaining cyclic channels at later nodes provided that they are available. Also when a packet is blocked for lack of any available output channels, the packet is not required to wait on any of these two groups of channels, rather it is routed on the first channel that becomes free. These two issues are important as they highly increase the flexibility of the generated routing algorithm. The theory develops the concept of *an extended channel dependency graph* that is formed by all the escape channels. The theory proves that the existence of a connected escape path that has no cycles in its extended channel dependency graph is a necessary and sufficient condition for the avoidance of deadlocks. Many adaptive and fully adaptive routing algorithm variations can and were developed using this model [4, 37].

3.3.1.3 Message Flow Model

The fundamental difference between the *message flow model* and the *channel dependency graph* model is that the latter represents the potential routing of packets from one channel to the next. The Message Flow model represents the actual flow of packets in the network. It avoids deadlocks by controlling the behavior of blocked packets in terms of which channels they are allowed to wait for when they are blocked so as not to introduce cyclic waits. It is therefore based on what is known as the *wait-for graph* technique. The model develops the idea of *deadlock-immune* channels. For any routing function, a channel is deadlock-immune for a particular packet if and only if once that packet occupies the channel, it will eventually release it. A channel that cannot be used by the packet, as specified by the routing function, is also considered to be deadlock-immune. Therefore, a channel is deadlock-immune if and only if it is deadlock-immune for all the routes provided by the routing function to a particular packet. The model proves that routing based on the idea of deadlock-immunity is a necessary and sufficient condition for developing deadlock-free routing algorithms. In similarity to the *extended channel dependency graph* model, the waiting channel concept used by this model is similar to the escape

channels subfunction provided by the latter. This also means that there is more routing freedom provided to packets that are not blocked, and therefore this model can achieve comparable highly flexible deadlock-free adaptive routing algorithms using minimal virtual channel resources [38].

3.3.2 Deadlock-recovery Algorithms

Deadlocks have been subjected to more in-depth analysis in recent papers. The outcome of these efforts indicate that deadlocks are extremely rare events that occur when the network is near or beyond its saturation point. Also in an effort to characterize deadlocks, these studies analyzed the effects of various factors on the probabilities of forming deadlocks. It was found that whenever more routing freedom was made available to packets in terms of virtual channel resources and routing path selections, then the probability of forming deadlocks by these packets diminishes exponentially [30, 31]. These important observations have paved the way for deadlock recovery techniques to be more widely accepted as a legitimate contender to the more traditional deadlock-avoidance techniques.

Deadlock-recovery algorithms allow packets to be routed on all the available channel resources without guarding against the possibility of deadlock. Once deadlocks occur, they are detected and a particular deadlock-recovery mechanism is invoked to resolve the deadlock situation using certain resources dedicated for that purpose. Fully adaptive routing algorithms that allow the use of all channel resources for routing packets without any restrictions, meaning that they employ deadlock-recovery techniques, are known as *true fully adaptive routing* algorithms [40, 41, 42]. Deadlock recovery algorithms are further classified as to how they deal with the packets involved in the deadlock situation.

3.3.2.1 Regressive (Abort-and-Retry) Recovery

In regressive algorithms, when the deadlock-recovery mechanism is invoked it selects one of the packets that is participating in the deadlock cycle for abortion. This includes clearing out all the buffers that the

packet holds, and aborting all internal crossbar connections within all the nodes that the packet currently passes through. This packet is then rescheduled for retransmission at a later time.

If the deadlock detection mechanism is based at the source node, then the source node sends a forward abort signal to all the nodes that the packet was routed through, and which reserve some resources for that packet. The node attempts to re-inject the packet into the network after some random delay. This mechanism requires the use of extra buffers that can store the entire packet per each injection and delivery channels in each node, so that packets can either be retransmitted or removed from its destination node after it has been partially received respectively. An example that uses this mechanism is the *compressionless routing* algorithm. The algorithm insures that a deadlocked packet can always be detected at the source node by padding all packets with *pad flits* until they reach their destinations [43].

If the detection mechanism is based at the node where the header of the packet resides, then a backward abort signal is sent upstream toward the source node of the packet in order to remove and clear all resources reserved by that packet. The source node then attempts to retransmit the packet at a later time. This mechanism requires the availability of packet buffers per each injection channel at each node. Also this technique assumes that the source node of a packet can always be determined [44].

3.3.2.2 Progressive Recovery

Progressive algorithms take a different approach in dealing with deadlocked packets. The deadlock recovery mechanism once again selects one of the packets involved in the deadlock cycle. This packet is then progressively advanced to its destination using either dedicated resources, or by temporarily reallocating resources from normal packets for the sake of the deadlock recovery process. Priority amongst all network packets is given to the deadlocked packet being progressed.

When the deadlock recovery mechanism uses special resources that are decoupled from the virtual channel resources that are used for normal adaptive routing, then it allows the adaptive algorithm to

provide increased routing flexibility using all the virtual channel resources, which is known as true fully adaptive routing. Depending on the frequency of deadlock occurrences in the network, it is important for these techniques to drain the deadlocked packets from the network quickly so as not to cause further congestion and performance deterioration. These techniques result in more efficient and simpler router design than regressive deadlock recovery techniques. Also they are more widely applicable to arbitrary interconnection network topologies and switching techniques [45, 46].

3.3.2.3 Backtracking Algorithms

These algorithms are based on packet backtracking and hardware modifications to the header of the packet in order to continuously route around congestion and possible deadlock cycles. An example of this is the Hyperswitch algorithm. Such algorithms might incur significant delays as they try all possible path permutations that can lead to the destination. Although pruning techniques can be used to reduce the search space, performance of these algorithms does not compare favorably to the other techniques presented here, and will not be considered further in this thesis [18, 39].

3.4 Routing Algorithm Examples

This section explains in more detail some of the well-known routing algorithms that were created throughout the literature, and may have been adopted for implementation within various commercial parallel machines. The examples discussed next cover the spectrum of algorithms as classified earlier.

3.4.1 Dimension-Ordered Routing Algorithm

This is one of the earliest and most popular routing algorithms proposed for wormhole networks. It is a deterministic algorithm proposed by Dally and Seitz in [33]. It avoids deadlock by using the channel

dependency graph model. Each packet is routed in one dimension at a time. Arriving at the proper coordinate in each dimension before proceeding to the next dimension. By enforcing a strictly monotonic order on the dimensions traversed, deadlock-free routing is guaranteed. The family of the Caltech Mesh-Routing chips (MRC), and the experimental Caltech Mosaic C fine-grain multicomputer implement the dimension-order routing algorithm [47, 48].

XY Routing Algorithm. When dimension-ordered routing is applied to two-dimensional meshes it produces the *XY* routing algorithm. Using this algorithm packets are sent first along the *X* dimension and then along the *Y* dimension. At most one turn is allowed, namely the one from the *X* dimension to the *Y* dimension.

Let (s_x, s_y) and (d_x, d_y) denote the addresses of a source and destination nodes respectively. Furthermore, let $(g_x, g_y) = (d_x - s_x, d_y - s_y)$. The *XY* algorithm can be implemented by placing g_x and g_y in the first two flits of a packet respectively. When the first flit of the packet arrives at a router, it is incremented or decremented, depending on whether it is greater or less than 0. If it is not equal to 0, the packet is forwarded in the same dimension and direction it arrived in. If the result equals 0 and the packet arrived on the *Y* dimension, the packet is delivered to the local processor. If the result equals 0 and the packet arrived on the *X* dimension, the flit is discarded and the next flit is examined upon arrival. If the second flit is 0, the packet is delivered to the local processor; otherwise, the packet is forwarded in the *Y* dimension.

E-cube Routing Algorithm. When dimension-ordered routing is applied to hypercubes or generally *n*-cube topologies it produces the E-cube routing algorithm. In here the packet header carries the destination node address *d*. Whenever a node *v* receives a packet, the algorithm computes $c = d \oplus v$ (XOR operation). If $c = 0$, then the packet is forwarded to the local processor. Otherwise the packet is forwarded on the outgoing channel in the *k*th dimension, where *k* is the position of the rightmost, or leftmost 1 in *c* depending on which dimension should the packet be routed in first [14, 33].

3.4.2 The Turn model

This model was proposed by Glass and Ni in [32] to provide a systematic approach to developing highly adaptive deadlock free algorithms based on the channel dependency graph model for deadlock-avoidance. The framework can also be used to develop livelock free, and minimal or nonminimal routing algorithms that are applicable to a variety of network topologies including n-dimensional meshes and k-ary n-cubes. A unique feature of the model is that it is not based on adding physical or virtual channels to networks, although it can be applied to networks with extra channels. Instead the model is based on analyzing the directions in which packets can turn in a network and the cycles that these turns can form, and then by prohibiting just enough turns to break these cycles. The 6 steps needed to implement the turn model are detailed below:

1) Partition the channels into different sets according to the direction in which they route packets. Wraparound channels are put in a separate set. If each node has v channels in a topological direction, they are treated as v distinct virtual directions, and are divided into distinct sets accordingly.

2) Identify the possible turns from one virtual direction to another. Omit 0-degree and 180-degree turns. A 0-degree turn is only possible when there are multiple channels in a topological direction.

3) Identify the cycles that the turns can form. Generally, identifying the simplest cycles in each plane of the topology is adequate.

4) Prohibit the minimum number of turns so that at least one turn is eliminated in each cycle. A useful approach is to first break the cycles in each plane and then check whether there are still more complex cycles to be removed.

5) Incorporate as many turns as possible involving the set of wraparound channels from step (1) without reintroducing cycles. At least one turn involving each wraparound channel can be incorporated.

6) Incorporate as many 0-degree and 180-degree turns as possible without reintroducing cycles.

For two-dimensional meshes, dimension 0 and 1 become x and y. The direction $-x$, $+x$, $-y$ and $+y$ become west, east, south, and north. The four directions permit eight 90-degree turns, which can form two simple cycles as illustrated in Figure 3.1a. The xy routing algorithm, discussed earlier, prohibits four of the turns as shown in Figure 3.1b. Although this restriction is sufficient to prevent deadlock, it does not allow any adaptivity while routing. Fortunately, deadlocks can be avoided by prohibiting fewer than four turns. Actually the turn model shows that only two turns need to be prohibited, one from each cycle. This allows adaptive routing to be performed. Depending on which two turns are prohibited distinguishes between the following different algorithm flavors.

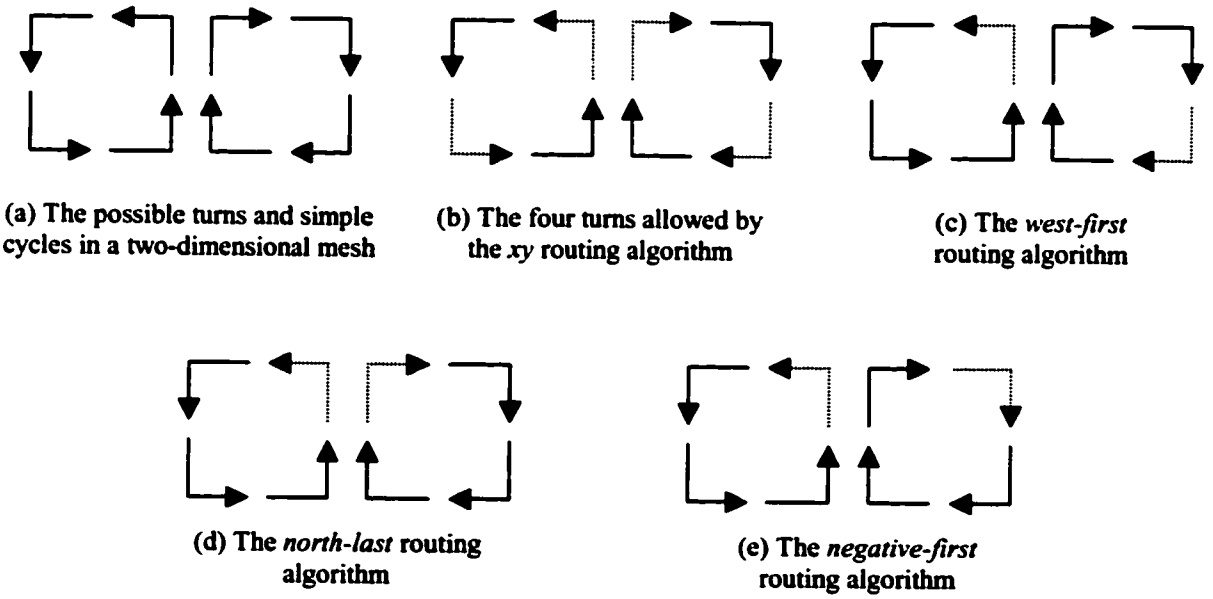


Figure 3.1. The Turn Model. Alternative methods of preventing cycles and the corresponding algorithms.

West-First Routing Algorithm. The two turns to the west are prohibited as shown in Figure 3.1c. Because of this a packet must start out in the west direction in order to travel west. The algorithm then accordingly routes a packet west first, if necessary, and then adaptively south, east, and north. Examples of both minimal and nonminimal routing using this algorithm on a 2D mesh are shown in Figure 3.2. For minimal routing, the algorithm is fully adaptive if the destination is on the right-handside (east) of the source; otherwise it is deterministic. For nonminimal routing the algorithm is adaptive.

There are other ways to prohibit cycles. If the two north turns are prohibited as in Figure 3.1d, or the two turns from a positive direction to a negative direction are prohibited as in Figure 3.1e, we obtain the *North-Last* Routing, and the *Negative-First* Routing Algorithms respectively.

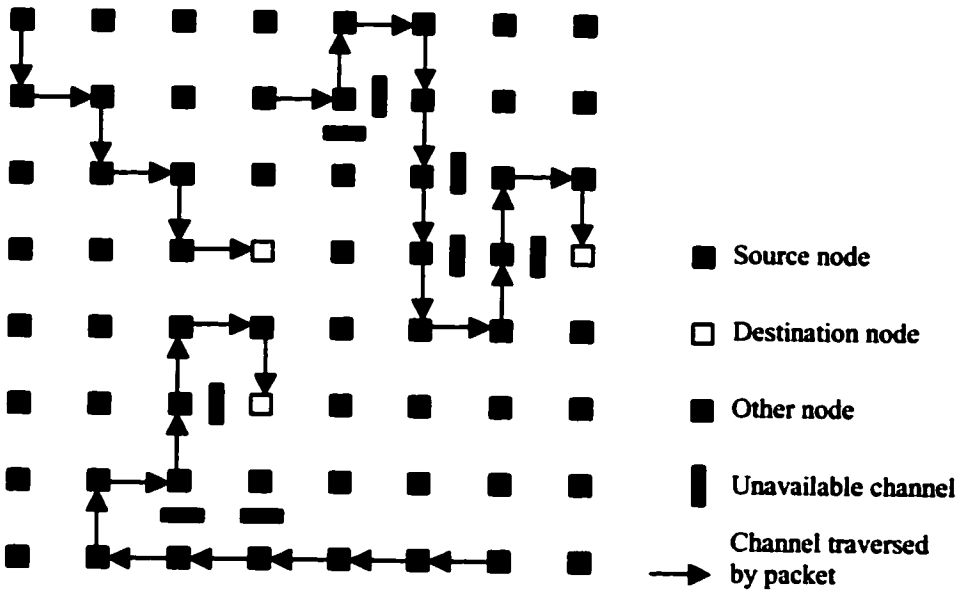


Figure 3.2. Example of *west-first* routing on an 8 x 8 2D mesh. Minimal and nonminimal paths are shown.

In the generalized n-dimensional mesh case, the analog of the west-first algorithm is the *all-but-one-negative-first* routing algorithm: route a packet first adaptively in the negative direction of all but one dimension ($n - 1$), and then adaptively in the other directions. The analog of the north-last algorithm is the *all-but-one-positive-last* routing algorithm: first route packets adaptively in the negative and the positive direction of dimension 0, and then adaptively in the other directions. The analog to the negative-first algorithm is also named the *negative-first* algorithm: route a packet first adaptively in the negative directions and then adaptively in the positive directions.

P-cube Routing Algorithm. This algorithm is for hypercube networks, which are special cases of both n-dimensional meshes and k-ary n-cubes. It is therefore an adaptation of the negative-first algorithm. Let S be the binary address of the source node for a packet, C be the binary address of the node the header flits currently occupy, and D be the binary address of the destination node. The p-cube routing algorithm has two phases. For minimal routing, the first phase routes the packet along any dimension i, for which $c_i = 1$ and $d_i = 0$. When there is no such dimension, the second phase routes the packet along any dimension i, for which $c_i = 0$ and $d_i = 1$. In nonminimal routing, the first phase can also route the packet along any dimension i, for which $c_i = 1$ and $d_i = 1$ [14, 32].

3.4.3 Planar-Adaptive Routing (PAR) Algorithm

This is a partially adaptive, deadlock-avoidance algorithm based on the channel dependency graph model. Adaptive routing can potentially improve network performance. Fully adaptive routing algorithms have much more routing freedom, which occurs at the expense of increasing hardware complexity. Fully adaptive routing requires large crossbar switch sizes, as they require each input buffer to be connected to all possible output buffers. Consequently crossbar switches grow quadratically with the number of dimensions n of the network. This can drastically reduce the clock speed for driving the network which counter effects the advantages of adaptive routing, as will be discussed in the *-Channels algorithm. *Planar-adaptive routing* proposed by A Chien in [49] restricts

the degree of adaptability to routing through a sequence of 2-dimensional planes until a packet reaches its destination, moving in one plane at a time. This significantly reduces the requirements for deadlock prevention. Using PAR, only three virtual channels are required per each physical channel for meshes and six virtual channels for tori regardless of the network dimension. Restricting adaptivity can lead to increased network clock rates, and therefore performance enhancements.

To illustrate the concept of this algorithm we consider an n-dimensional mesh as depicted in Figure 3.3. A fully adaptive algorithm would allow a packet traveling from the current to the destination nodes shown in Figure 3.3a to be routed in the m-dimensional cube defined by the two nodes. PAR on the other hand restricts routing the packet in one plane at a time. So the packet is routed in plane A_0 first, then A_1 as shown in 3.3b, or in A_0, followed by A_1, and then A_2 as shown in 3.3c. Within each plane a packet can be routed on all possible paths in that plane, and depending on the current offsets of the coordinates of the packet in relation to the dimensions in that plane.

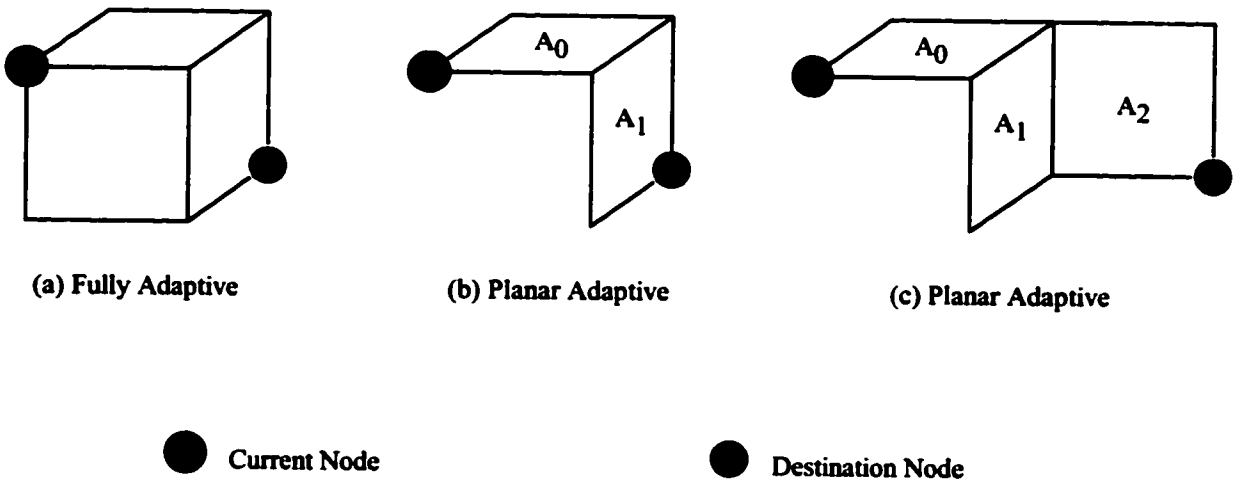


Figure 3.3. Planar Adaptive Routing (PAR)

Each plane A_i is composed of two dimensions, d_i and d_{i+1} for a total of $n-1$ adaptive planes in an n-dimensional network. Each two consecutive planes share one dimension. For example planes A_i and A_{i+1} share dimension d_{i+1}, and whenever the offset in d_i becomes zero, then routing can be switched to the A_{i+1} plane. But if the offset in d_{i+1} becomes zero while still routing in plane A_i then there is no adaptivity left in the A_{i+1} plane and routing in that plane will follow a straight line, or can be skipped altogether for the next plane.

To illustrate how PAR guarantees deadlock of freedom we consider a mesh network, which requires two virtual channels. Let $d_{i,j}$ be the set of j virtual channels in dimension i of the mesh network. This set is broken into two virtual channel subsets, one in each positive and negative directions of the network. Let $d_{i,j}+$ be the positive direction virtual channels, while $d_{i,j}-$ represent those in the negative direction. A plane A_i is then composed of the following sets of virtual channels:

$$A_i = d_{i,2} + d_{i+1,0} + d_{i+1,1}$$

The set of virtual channels in plane A_i is now divided into two separate virtual networks, one in the increasing direction, and the other in the decreasing direction of that plane. Let A_{i+} represent the increasing network, while A_{i-} represents the decreasing network, then:

$$A_{i+} = d_{i,2+} + d_{i+1,0} \quad \text{and,}$$

$$A_{i-} = d_{i,2-} + d_{i+1,1}$$

The routing algorithm then routes packets that need to travel in the positive direction of d_i in the A_i+ network, and packets that are traveling in the negative direction are routed in the A_i- network. By preventing cycles amongst these two subnetworks in a particular plane, freedom of deadlock is guaranteed.

The performance of the PAR algorithm was shown to be able to outperform some fully adaptive routing algorithms, whenever the same number of virtual channels are allocated to both. The extra virtual channels beyond what PAR needs are used as virtual lanes for extra adaptivity. Also it was shown that PAR performs better than deterministic routing algorithms under nonuniform traffic patterns, and performs comparably to them under the uniform traffic pattern [36, 49, 50].

3.4.4 Linder-Harden Algorithm

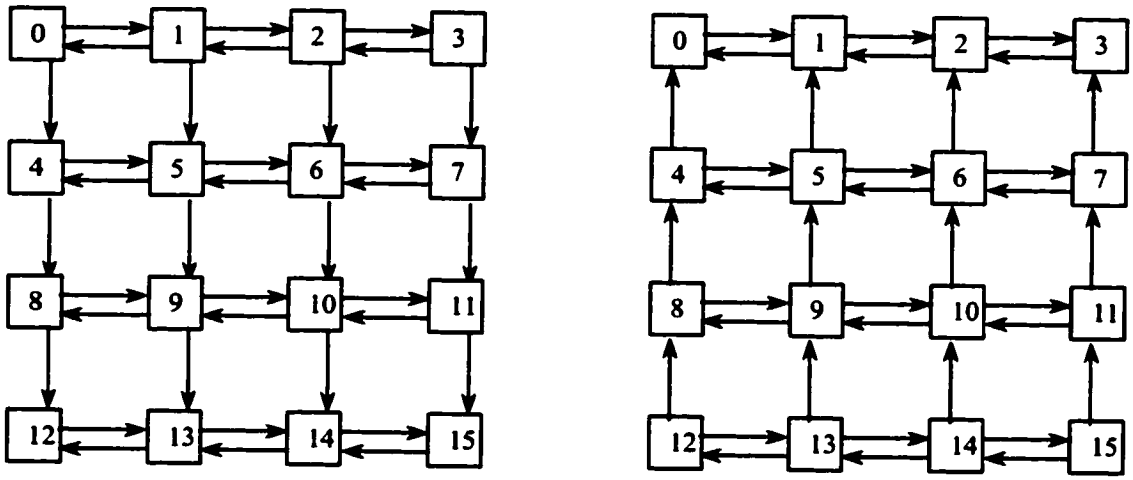
This scheme proposed by Linder and Harden in [17] is a minimal, fully adaptive, and fault tolerant deadlock-avoidance algorithm. The concept used here is an extension of the channel dependency graph model into *virtual networks*, and allows for the simple creation of deadlock-free routing algorithms. The basic principle here is to split the physical network into a disjoint set of virtual networks, each having its own subset of virtual channels. Various packets are routed through different sets of virtual networks depending on the direction they need to follow in order to reach their destination. By structuring the virtual networks in a way that avoids cyclic turns within each virtual network, and amongst all the sets of virtual networks, deadlock-free routing is obtained.

This concept can be used for the n-dimensional torus and mesh networks. An n-dimensional mesh requires 2^n - 1 virtual networks. Each virtual network has two directions in the 0-dimension and only one in the remaining $n-1$ dimensions. The number of virtual channels required per node then is $n2^n$ - 1 plus an extra virtual channel in the 0-dimension for a total of $(n+1)2^n$ - 1 virtual channels.

For example, in a 2D mesh, an additional pair of channels is added to the X dimension. The network is then partitioned into two virtual networks 0, and 1. Each virtual network has a pair of channels in the X dimension and a unidirectional channel in the Y dimension. If the destination node is to the right of the source, that is $d_X > s_X$, then the packet will be routed through virtual network 0. If $d_X < s_X$, virtual network 1 is used. If $d_X = s_X$, the packet can be routed through either of the subnetworks. Refer to Figure 3.4 for an illustration. In this case, the algorithm is referred to as the *double-X*

algorithm.

The concept can be extended to the more general k-ary n-cube networks as well. When wraparound channels exist, in the torus network case, each virtual network is further split into several levels; each level again has a subset of the virtual channels. Packets are routed the same way as in the mesh network, except that now whenever the packet is routed along a wraparound channel, then it is routed through another level of that particular virtual network. Since n + 1 levels are required per virtual network to accommodate all the turns a packet can make in the whole network for minimal routing. The total number of virtual channels required are $(n+1)2^n - 1$ for all dimensions, except the 0-dimension where $(n+1)2^n$ virtual channels are needed.



(a) Virtual Network 0

(a) Virtual Network 1

Figure 3.4. The Linder-Harden Algorithm

The main and obvious drawback of this algorithm is that the number of additional virtual channels increases exponentially with n, making it impractical for large networks. The other main drawback is that the routing algorithm is not as adaptive and fault tolerant as possible, given the large number of virtual channels it requires [14, 17, 51, 52, 53].

3.4.5 Dimension-reversal Algorithm

This nonminimal adaptive deadlock-avoidance algorithm proposed in [15] can be applied to any k-ary n-cube including mesh networks. The algorithm is based on controlling the number of *dimension reversals* (*DR*) a packet is allowed to take in order to break dependency cycles. The *DR* number is defined as the number of times a packet has been routed from a channel in a any dimension p, to another channel in a lower dimension q, where $q < p$. The idea capitalized on by this algorithm is that the number of additional virtual channels can be reduced if minimal routing is not required.

The *static dimension reversal* algorithm is based on the channel dependency graph model for deadlock-avoidance. It uses r pairs of virtual channels that are added between two adjacent nodes. The network is then partitioned into r subnetworks. The class-i ($0 \le i \le r-1$) subnetwork consists of all the ith pair channels. The packet header carries an additional class field c initially set to 0. Packets with $c < r-1$ can be routed in any direction in the class-c network; thus the route may be nonminimal. Each time a packet is routed from a high to a low-dimensional channel, or reverse dimension ordering, the c field is increased by 1. Once the c value has reached $r-1$, the packet must use the deterministic dimension-ordered algorithm for the remainder of the path. The parameter r limits the number of times reverse ordering can take place, and hence dictates the degree of adaptability of the routing algorithm. This algorithm makes inefficient use of virtual channels as a packet may be blocked waiting for a virtual channel in its class, while other virtual channels of the same physical channel remain idle.

The *dynamic dimension reversal* algorithm overcomes the limitations of the static algorithm by permitting packets to use any available virtual channel. It prevents deadlocks by eliminating cycles in

the packet *wait-for graph* rather than the channel dependency graph. The channels are divided into two nonempty classes: adaptive and deterministic. Packets originate in the adaptive channels, where they can be routed in any direction with no limit on the number of dimension reversals. However a packet with $c = p$ is not allowed to wait on a channel currently used by a packet with $c = q$ whenever $p \geq q$. A packet that reaches a node where all permissible output channels are occupied by packets whose values of c are less than or equal to its own must switch to the deterministic class of channels using the dimension-order algorithm. This algorithm may require a more complex router to perform the extra logic, but it takes the first step toward producing more flexible adaptive routers by removing the restriction of acylic dependencies. The algorithm allows cyclic dependencies amongst packets as long as they do not wait for channels in a cyclic manner.

Both of these algorithms require a header field in each message to record the number of dimension reversals performed. The number of virtual channels assumed in [15], 16 per physical channel is also quite large [14, 15, 53].

3.4.6 The *-Channel Algorithm

The *-Channel is a minimal and fully adaptive routing algorithm for n-dimensional torus networks proposed by Berman et al. in [52]. It avoids deadlock by using the extended channel dependency graph model. The main idea is simple. There are basically two types of virtual channels, *star* channels c^*, and *nonstar* channels c. Messages will move through the star channels according to the *dimension-order* oblivious routing. The nonstar channels will be used when taking any of the transitions that would not be allowed by the dimension-order algorithm. This guarantees freedom of deadlock, while allowing full adaptivity. A packet starts transmission in a dimension i. If dimension i is the highest dimension which the packet needs to be transmitted in for all of its minimal paths, then it goes through c^*i in this dimension, otherwise it uses ci. Each star channel c^*i is partitioned into two levels: $c^*i,0$ and $c^*i,1$. If the packet has taken a warp-around link along dimension i, then it uses $c^*i,1$. Otherwise, it uses the

c^*i,0 virtual channels.

The *-Channel requires a moderate amount of resources for fully adaptive routing. It has the lowest known channel requirements for fully adaptive minimal routing. Only five virtual channels are needed per bidirectional link in n-dimensional tori. The number of virtual channels can actually be made to three for one of the dimensions. More importantly the number of virtual channels needed per link does not depend on n or the size of the network. In [52], it was shown that the performance of this algorithm was better than other adaptive and oblivious algorithms, even for the random traffic distribution. This result is disputable, because the study did not take into account the increased router setup delay and flow control latency due to the mainly larger crossbar switch size needed for full adaptive routing. In a cost and speed model study [34], the *-Channel algorithm was compared against other deterministic (dimension-order), and adaptive (PAR, Turn model) algorithms, taking these two metrics into account. The study concluded that the *-Channel algorithm was the most costly alternative with higher setup delay, and flow control latencies. Consequently, latency for the *-Channels was found to be significantly worse than the other algorithms. This will be explained in more details in section 3.5 of this chapter [52, 53, 54].

3.4.7 The Duato Protocol

The Duato protocol is a framework for developing maximally adaptive routing algorithms. The framework provides deadlock-free routing algorithms based on the extended channel dependency graph model. The model results in developing maximally adaptive algorithms using a minimal set of virtual channel resources. The central idea here is to provide a routing subfunction that is connected and deadlock-free that operates on a subset of the virtual channels of the network. This routing subfunction provides the escape channels that blocked packets can always use whenever they are blocked. The remaining virtual channels can be used adaptively without any restrictions. A fully adaptive algorithm that is derived using this model for 2D meshes is presented next, and will be used later in the simulation

study.

The routing subfunction here is to be based on the deterministic *Dimension-ordered* routing, or more specifically the *XY* routing algorithm. This algorithm requires only one virtual channel for deadlock-free routing in 2D meshes. Packets using this routing subfunction are routed in the *X* dimension first, then the *Y* dimension. The escape virtual channel is presented as one of the alternative paths for the packet in a way that obeys the *XY* routing algorithm restrictions. The remaining virtual channels are utilized adaptively without any restrictions and are supplied by the routing function. It is important to note here that in each step of the routing decision, all virtual channels from both routing functions (adaptive and escape channels) are presented as possible paths to the packet for increased routing flexibility. Whenever a packet uses the escape channel, it is free to go back to the adaptive virtual channels set at later nodes. Also when the packet is blocked for lack of available channels, it does not wait on any particular set of channels, rather it is free to use the next virtual channel that becomes available [4, 38].

3.4.8 DISHA Algorithm

This scheme belongs to the class of *true fully adaptive routing* (TFAR), *progressive* deadlock-recovery algorithms. Routing is allowed on all available virtual lanes without regard to deadlock. Once deadlocks occur, they are detected and the *Disha* deadlock recovery mechanism is invoked to break the deadlock cycle in a progressive fashion. The main idea is to allocate a central buffer, called the deadlock buffer (*DB*), in each node. This buffer is connected to the router's crossbar via an input port, and is accessible by all neighboring nodes. When a deadlock is detected, one of the packets involved in the deadlock formation is transferred to the deadlock buffer of the next node along its path. From that point on, the packet continues to advance using only this "floating" lane composed of all the deadlock buffers throughout the network until it reaches its destination. Deadlocks on the floating lane are prevented by using either a sequential or concurrent deadlock recovery mechanism. In sequential

access, a token is needed to synchronize access to the deadlock buffers. While in concurrent access, the deadlock buffers themselves are structured in a way that prevents deadlock, much like the way deadlock-avoidance algorithms work. This guarantees that more than one deadlocked packet can utilize the deadlock lane concurrently without any conflicts that might lead to deadlocks. Sequential Disha deadlock recovery will be considered here, and as described in [45].

To illustrate how the algorithm works, we revisit a single-cycle deadlock formation as displayed in Figure 3.5 below. In the figure four packets are involved in the deadlock. Source and destination nodes for all packets are denoted as $S(x)$, and $D(x)$, where x is the number of the packet. Also three routers R_1, R_2, and R_3 are marked and will be considered in more detail next.

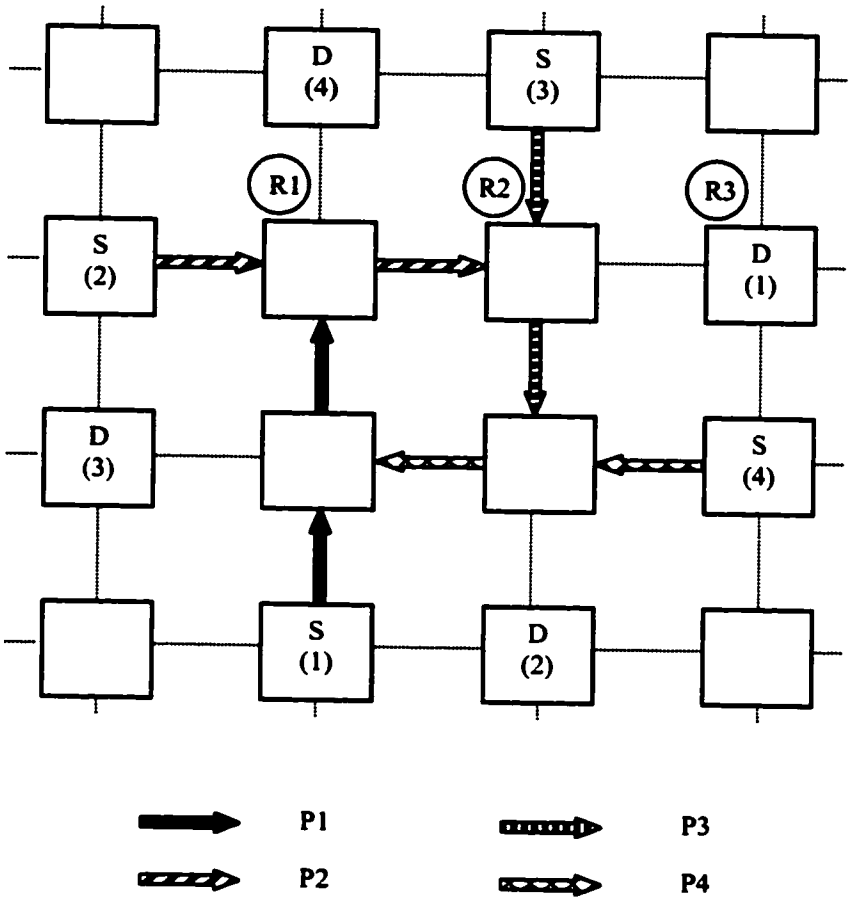


Figure 3.5. Single deadlock configuration with one virtual channel

This deadlock situation is depicted in more detail in Figure 3.6 below. In the figure, the internal architecture of the Disha router is shown. This simple example considers a 2D mesh with a single virtual channel per physical channel. Routers R_1 and R_2 are as marked on Figure 3.5. Also note the deadlock buffer (*DB*) that is connected via an extra input port to the crossbar. The processor of the node is also shown with a single injection, and delivery channels. As can be seen, packet 1 arriving into R_1 from the positive Y dimension is stalled waiting on the output buffer of the positive X dimension

occupied by P_2. Further upstream, P_2 is also stalled in R_2 waiting on the output buffer occupied by P_3. P_3 is also waiting on resources held by P_4. The latter is again stalled waiting on P_1 completing the deadlock cycle, which will not be resolved without manual intervention by the deadlock logic. The deadlock detection mechanism in Disha is based on a timeout mechanism. If a packet cannot be successfully routed for a predetermined threshold timeout value, then the packet is presumed to be involved in a deadlock formation. At this point the Disha deadlock recovery mechanism is triggered, and will be explained next.

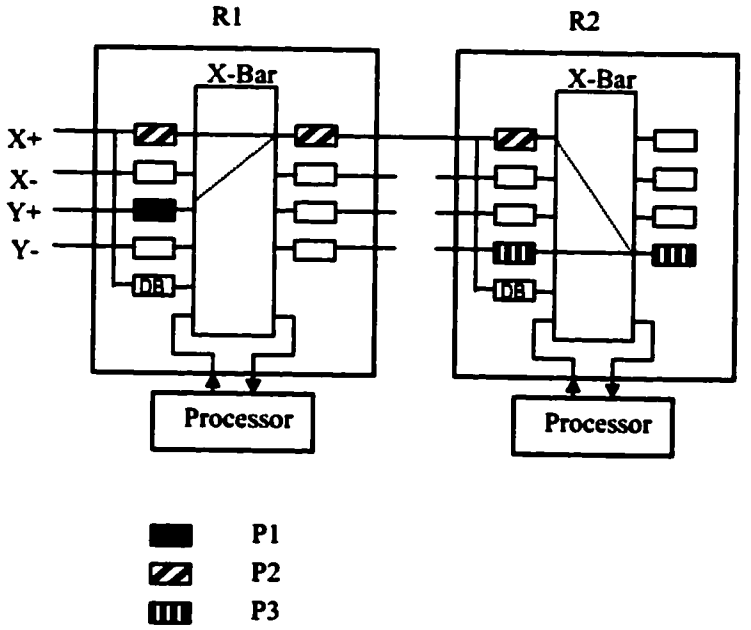


Figure 3.6. Detailed deadlock in the Disha router

In Figure 3.7 the progressive action of the Disha deadlock recovery is shown. When the deadlock recovery mechanism is activated in R_1, it clears the output buffer, and physical channel bandwidth needed by P_1 from the other packets, P_2 in this case. It then asserts the corresponding status line for R_2

to receive the header of P_1 and places it in its deadlock buffer. R_2 now receives the header in its deadlock buffer bypassing the input buffers occupied by P_2. R_2 continues the same process using the deadlock buffers throughout all subsequent nodes until the packet reaches its destination. When P_1 arrives at the deadlock buffer in R_3, it is directed to the delivery channel of the local processor for consumption. When P_1 is delivered, the deadlock cycle is broken and the other packets participating in the deadlock can also resume their routing process to their destination using the released resources. Various registers and crossbar reconfiguration is needed in order to resume the routing of certain suspended packets, such as P_2 in this case.

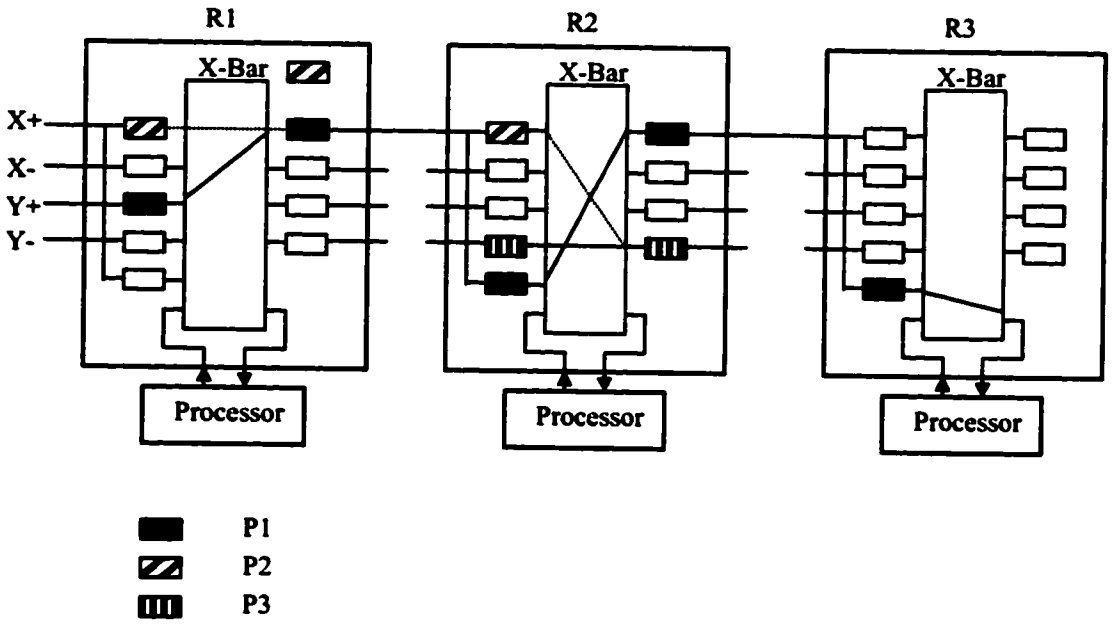


Figure 3.7. Detailed deadlock recovery in the Disha router

This method requires the allocation of a central deadlock buffer that is connected to the crossbar through an input port at each node, a hardware token signal wires connected to all nodes, An extra status

line to select the deadlock buffer (treated just as another virtual channel) through the virtual channel controller, and a crossbar reconfiguration buffer used to temporarily store any broken connection inside the crossbar while forwarding the deadlocked packet [45].

Requirements for simultaneous concurrent Disha deadlock recovery are as follows. A network topology that has Hamiltonian paths, or has an embedded spanning tree is required. Two deadlock buffers are also needed per node. Nodes are labeled according to their position in the Hamiltonian path. Packets with higher destination labels than the current node are routed on one set of deadlock buffers, while packets with lower destination labels than the current node are routed on the other set in order to avoid deadlock on the deadlock buffer lanes [4, 55].

3.5 Cost Model

Most performance studies of the routing algorithms in the literature are based on achieved network throughput and utilization figures that assume constant router delays and network clock cycles. This method ignores the implementation complexity of the routers and its underlying effect on the achievable network clock rates. The cost model proposed in [34] attempts to evaluate the router architecture for various routing algorithms based on the implementation complexity of those routers. The study develops a parametric cost and speed model for a family of routers based on a canonical router architecture. This model allows the direct comparison of the different routing algorithms based on the cost of their hardware resources requirements, and ultimately their realizable router speed. This model is especially useful in the evaluation of different adaptive routing algorithms, as they generally require the utilization of more expensive hardware resources. The extra hardware resources needed may not be highly utilized up to their full potential, such as the case in the deadlock-avoidance adaptive routing algorithms.

3.5.1 Canonical Router Architecture

The basic router architecture adopted in this model is shown in Figure 3.8 below. The data or packets move through this router from left to right, while the flow control signals are propagated in the opposite direction to regulate the flow of data through the network.

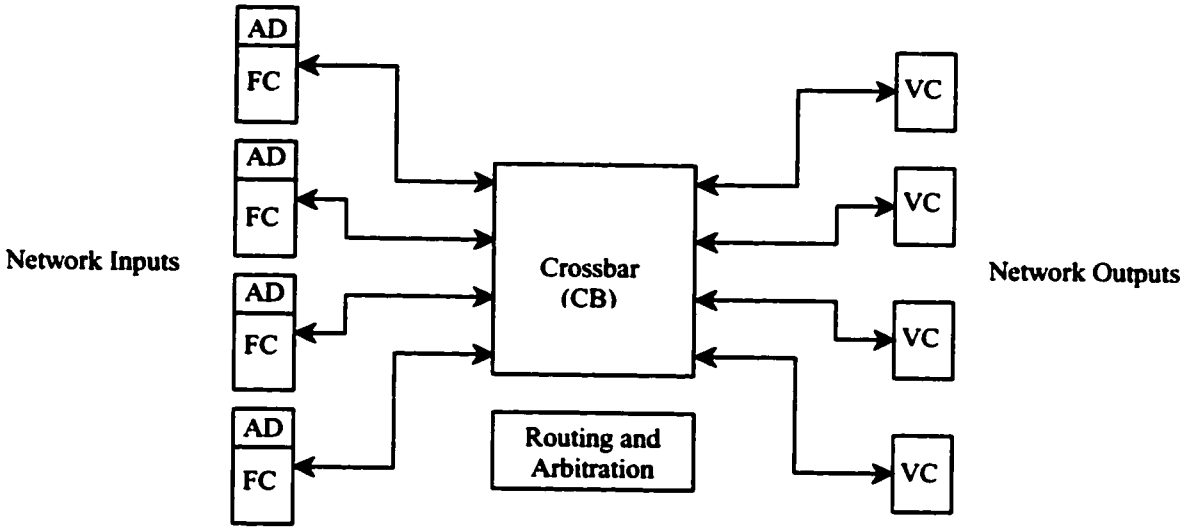


Figure 3.8. Canonical router architecture.

The basic modules of this canonical router architecture are:

- **Crossbar (*CB*).** Implements the basic switching function from input to output ports. The crossbar switch can either be complete, where all inputs are connected to all outputs. Otherwise the crossbar can be partitioned into a set of smaller crossbars for better performance.
- **Flow Control unit (*FC*).** Implements the flow control protocol between adjacent routers, which is responsible for halting and resuming the progress of packets depending on buffer space

availability.

- **Address Decoder (*AD*).** This unit is responsible for examining the header of incoming packets, determine the set of possible paths according to a particular routing algorithm, and update the headers accordingly.
- **Routing Arbitration logic (*RA*).** Selects the appropriate path for a packet according to a particular routing and selection functions, sets the appropriate input and output switch connections for that path. Also, the unit performs arbitration amongst competing incoming packets for the same output paths.
- **Virtual Channel controller (*VC*).** Multiplex the virtual channels on a particular physical channel. Controller complexity increases superlinearly according to the number of virtual channels required.

3.5.2 Intra-router Performance Measurements

This model concentrates on *intra-router*, rather than *inter-router* performance measures. The two main measures of intra-router performance are the *routing latency*, and the *flow control latency*.

The routing latency is the time consumed while establishing a connection through the router for a new packet. It is also known as the *header latency*, or *path setup delay* and can be broken down into the following components:

1) Address decoding, and generating the connect request (T_{AD})
2) Arbitration (T_{ARB})
3) Updating the header with path selection (T_{SEL})
4) Switching the data across the crossbar (T_{CB})
5) Virtual channel controller delay (T_{VC})

The *flow control*, or *data through* latency ultimately determines the flit transmission rate on the network channels. The channel bandwidth is therefore dependent on the flow control unit (*flit*), and the time it takes to perform a flow control operation. The flow control operation is mainly concerned with the delay for the data flits and can be broken down into the following components:

1) Flow controller delay (T_{FC})
2) Forward crossbar switching delay of data (T_{CB})
3) Virtual channel controller delay (T_{VC})

3.5.3 Parametric Cost Model

The parametric cost model developed in the study is based on the hardware complexity of each of the router modules described above. Since most routing algorithms can be implemented using the canonical router architecture, these routing algorithms can be directly compared to each other based on the differences in the individual hardware modules used within their routers. Formulas for the gate count complexity and delay for each of the modules is listed in Table 3.1 below [34].

TABLE 3.1 Gate Counts and Delays for the Canonical Router Components. *P* is the number of input or output ports for the crossbar. *F* is the routing freedom available, and is usually equal to *P*. *V* is the number of virtual channels per physical channel

Module	Parameter	Gate Count	Delay
Crossbar	P (ports)	$O(P^2)$	$c_0 + c_1 * \log_2 P$
Flow Control Unit	None	O(1)	c_2
Address Decoder	None	O(1)	c_3
Routing Decision	F (freedom)	$O(F^2)$	$c_4 + c_5 * \log_2 F$
Header Selection	F (freedom)	O(F)	$c_6 + c_7 * \log_2 F$
VC Controllers	V (# of VCs)	O(V)	$c_8 + c_9 * \log_2 V$

In order to evaluate the different routers using this model, it is instantiated for a 0.8-micron CMOS gate array technology. This solves the constants in the model. The values of the constants are shown in Table 3.2 below [34].

TABLE 3.2 Components Delay Constants for a 0.8-micron CMOS Technology

Constant	Value (nanoseconds)
c_0	0.4
c_1	0.6
c_2	2.2
c_3	2.7
c_4	0.6
c_5	0.6
c_6	1.4
c_7	0.6
c_8	1.24
c_9	0.6

Following sections use this parametric cost model to evaluate several routing algorithms. This sheds light on the internal router architecture of various routing algorithms, and how the router design can be adapted to the routing algorithm for optimum performance on the hardware level. The routers shown are for the 2-dimensional network, but can be easily extended to any n-dimensional network.

3.5.4 Dimension-order Router

The setup delay in the dimension-order router is composed of decoding the address, updating the header, a simple routing decision is made (either continue in the current dimension or advance the packet to the next dimension), establish the connection through the crossbar, and switch the data through it. Because the dimension-order router routes a packet in one dimension at a time before proceeding to the next dimension, the router design can be optimized to contain a crossbar for each dimension. Each subcrossbar has only three inputs, $D+$, $D-$, and the connection from the higher dimension. It only has three outputs as well, $D+$, $D-$, and the connection to the lower dimension. Each subcrossbar becomes specialized in routing packets in one dimension, and is therefore efficient and fast. The router architecture is illustrated in Figure 3.9.

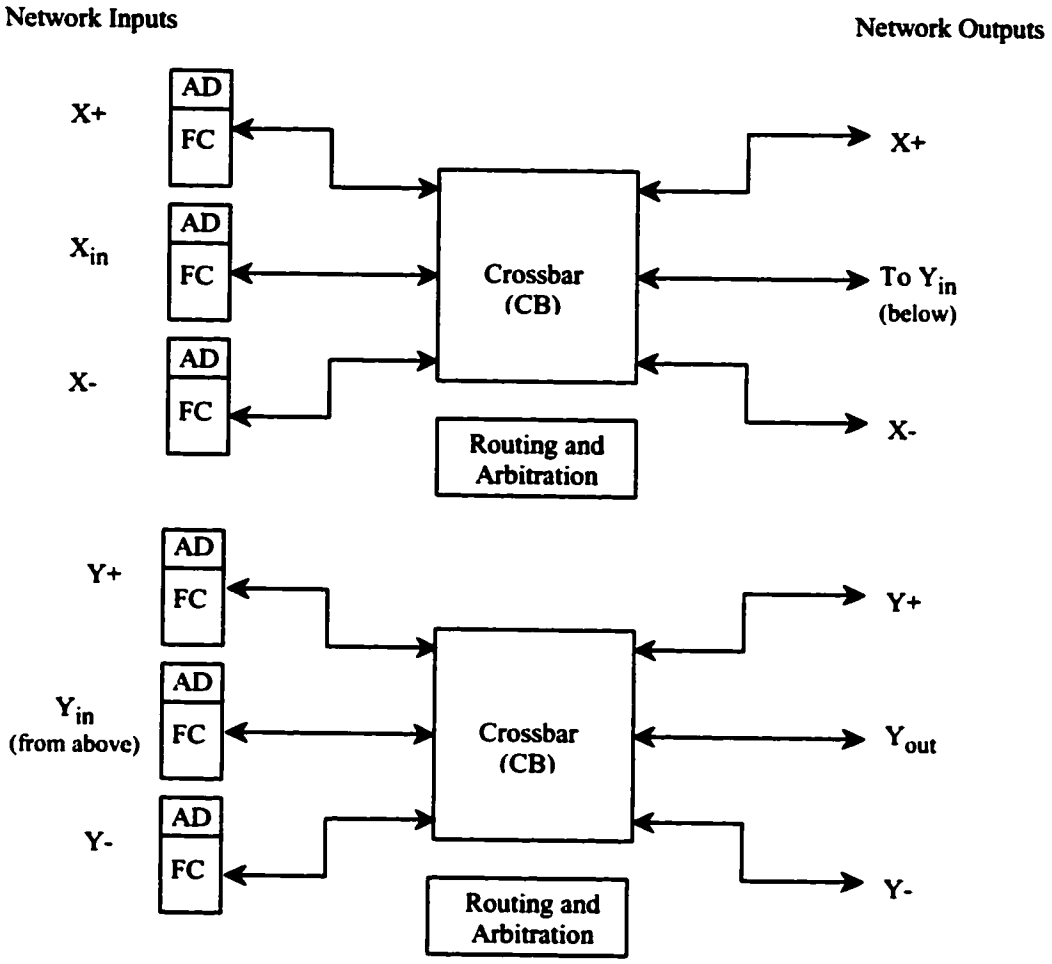


Figure 3.9. Block diagram of the dimension-order router.

The equation for the setup delay for the dimension-order router can then be described as:

$$T_{DimOrder} = T_{AD} + T_{ARB} + T_{CB}$$

For a 2-dimensional network, $P = 3$, $F = 3$, and $V = 0$. Substituting from Table 3.1 we have the following expression:

$$T_{DimOrder} = c_3 + c_4 + c_5 * \log_2 F + c_0 + c_1 * \log_2 P$$

$$T_{DimOrder} = c_3 + c_4 + c_5 * \log_2 3 + c_0 + c_1 * \log_2 3$$

Solving this equation using Table 3.2 yields:

$$T_{DimOrder} = 5.6\ ns$$

The flow control delay is calculated using:

$$T_{fc\text{-}DimOrder} = T_{FC} + T_{CB}$$

$$T_{fc\text{-}DimOrder} = c_2 + c_0 + c_1 * \log_2 P$$

$$T_{fc\text{-}DimOrder} = 3.55\ ns$$

The dimension-order router has very simple hardware as shown by the setup and flow control delays. The router can therefore operate at higher clock rates in comparison to the adaptive routers. The advantages in speed are lost however, if virtual lanes are added to the router.

3.5.5 Planar-Adaptive Router

PAR attempts to limit the hardware cost associated with adaptive routing by limiting adaptivity to two dimensions at a time. This can in turn be reflected in the router design by using different subcrossbars for each plane, each subcrossbar is a 4x4 switch regardless of the network dimension. The router design

is shown in Figure 3.10.

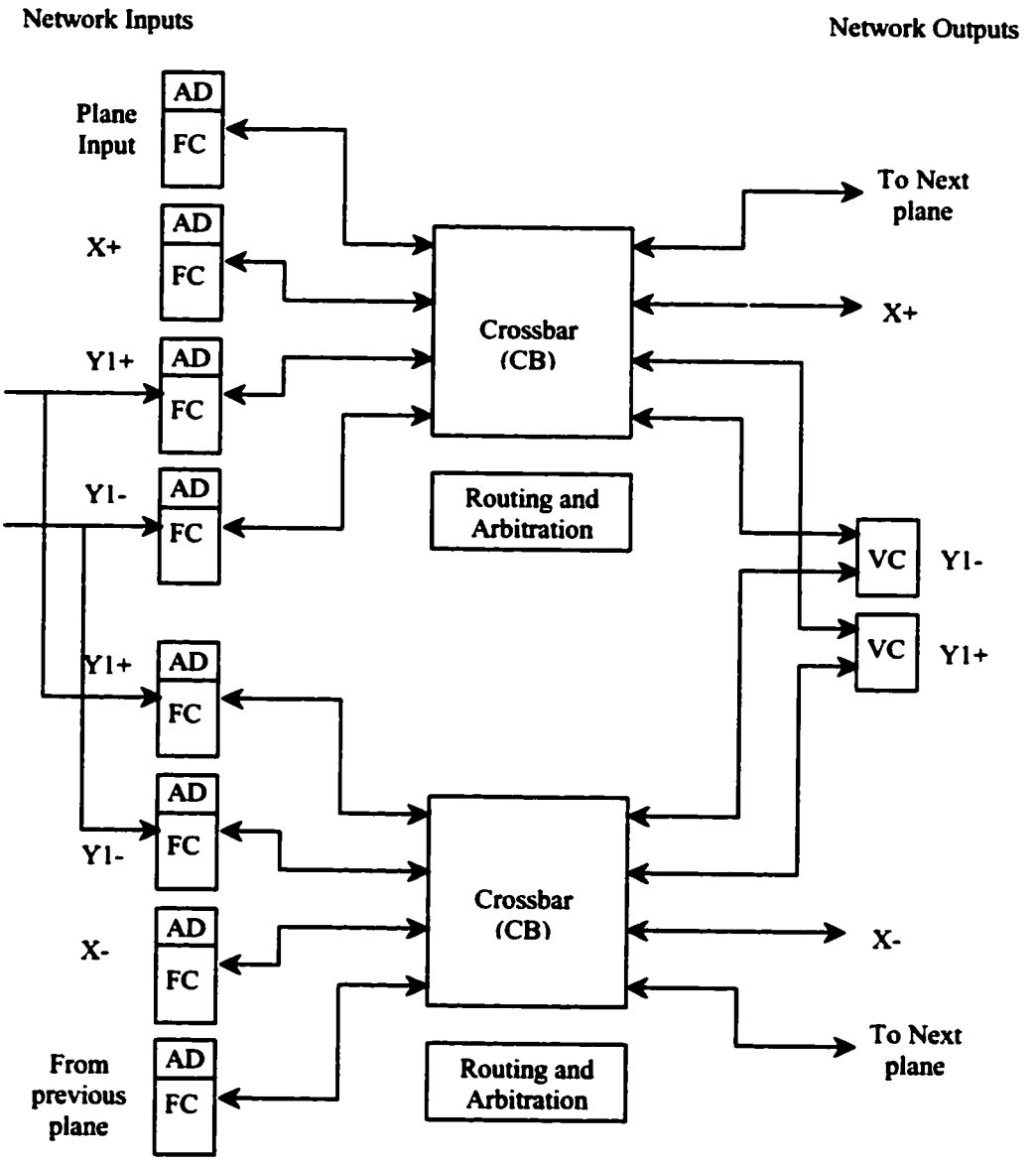


Figure 3.10. Block diagram of the Planar-Adaptive router.

Although this is a controlled adaptive router, certain delays characteristic of all adaptive routers are present. Namely, the delay needed by the address decoders in order to examine the headers, then update them according to the path selected by the routing algorithm. Also this algorithm requires the utilization of three virtual channels for deadlock avoidance. The critical path for the setup delay while routing a packet is then composed of:

$$T_{PAR} = T_{AD} + T_{ARB} + T_{SEL} + T_{CB} + T_{VC}$$

Using Table 3.1 we substitute for the expressions:

$$T_{PAR} = c_3 + c_4 + c_5 * \log_2 F + c_6 + c_7 * \log_2 F + c_0 + c_1 * \log_2 P + c_8 + c_9 + * \log_2 V$$

For a 2-dimensional network, $P = 4$, $F = 4$, and $V = 3$,

$$T_{PAR} = c_3 + c_4 + c_5 * \log_2 4 + c_6 + c_7 * \log_2 4 + c_0 + c_1 * \log_2 4 + c_8 + c_9 + * \log_2 3$$

Substituting the constants from Table 3.2 we get:

$$T_{PAR} = 10.9\ ns$$

The flow control latency is given by:

$$T_{fc\text{-}PAR} = T_{FC} + T_{CB} + T_{VC}$$

$$T_{fc\text{-}PAR} = c_2 + c_0 + c_1 * \log_2 P + c_8 + c_9 * \log_2 V$$

Substituting for the variables and the constants:

$$T_{fc\text{-}PAR} = 6.15\ ns$$

The setup and flow control latencies for the PAR router are almost double those of the dimension-order router. Adaptive routing increases those latencies on account of the introduced header selection, and virtual channel multiplexing delays.

3.5.6 Turn Model Router

The Turn model is a collection of adaptive algorithms that avoid deadlock by preventing certain basic turns, and it does not require the use of virtual channels. The *negative-first* routing algorithm is selected. This algorithm routes adaptively in all of the negative directions first, then route adaptively in all of the positive directions. The router for this algorithm requires the use of a single crossbar that connects all input to all output ports. The router diagram is shown in Figure 3.11.

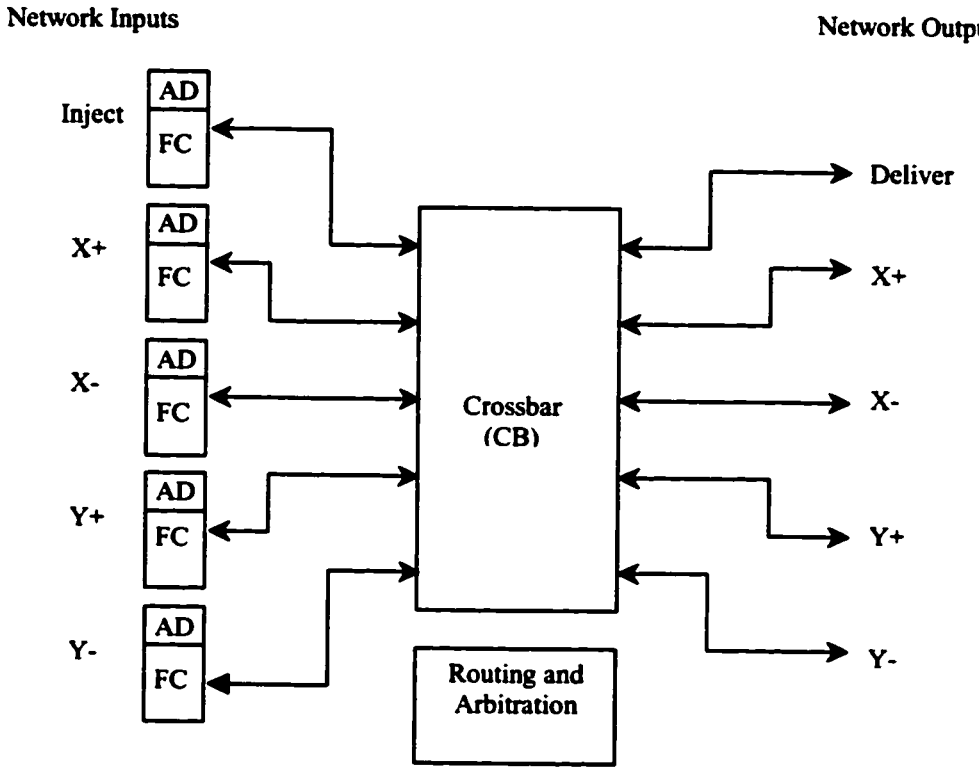


Figure 3.11. Block diagram of the Turn model router.

The setup delay reflects using a large crossbar, and a larger arbitration circuits. The setup delay is derived as in other adaptive routers, but without the use of virtual channels:

$$T_{Turn} = T_{AD} + T_{ARB} + T_{SEL} + T_{CB}$$

Substituting these components from Table 3.1:

$$T_{Turn} = c_3 + c_4 + c_5 * \log_2 F + c_6 + c_7 * \log_2 F + c_0 + c_1 * \log_2 P$$

In here $F = P = 2D + 1$, where D is the dimension of the network. The number of dimensions is doubled because we have one large crossbar for the two dimensions. The extra port is the network input and output port:

$$T_{Turn} = c_3 + c_4 + c_5 * \log_2 (2D + 1) + c_6 + c_7 * \log_2 (2D + 1) + c_0 + c_1 * \log_2 (2D + 1)$$

Assuming a 2-dimensional network:

$$T_{Turn} = 9.1\ ns$$

The flow control latency is similarly given by:

$$T_{fc\text{-}Turn} = T_{FC} + T_{CB}$$

$$T_{fc\text{-}Turn} = c_2 + c_0 + c_1 * \log_2 P$$

$$T_{fc\text{-}Turn} = c_2 + c_0 + c_1 * \log_2 (2D + 1)$$

Again assuming a 2-dimensional network:

$$T_{fc\text{-}Turn} = 4.0\ ns$$

The latencies of the Turn model are larger than those of the dimension-order router, but they are less than those of the PAR router because virtual channels are not required by the Turn model router. In higher dimension networks the advantages of the Turn model over PAR start to disappear because of the larger crossbars and routing arbitration units that are needed.

3.5.7 *-Channel Router

The *-Channel algorithm is a fully adaptive deadlock-avoidance router that uses a moderate number of virtual channels. The algorithm requires a single large crossbar of size $(4D + 1) \times (4D + 1)$, or 9 x 9 for a 2-dimensional network. The crossbar has an input port for each virtual channel used. The router architecture is show in Figure 3.12 below.

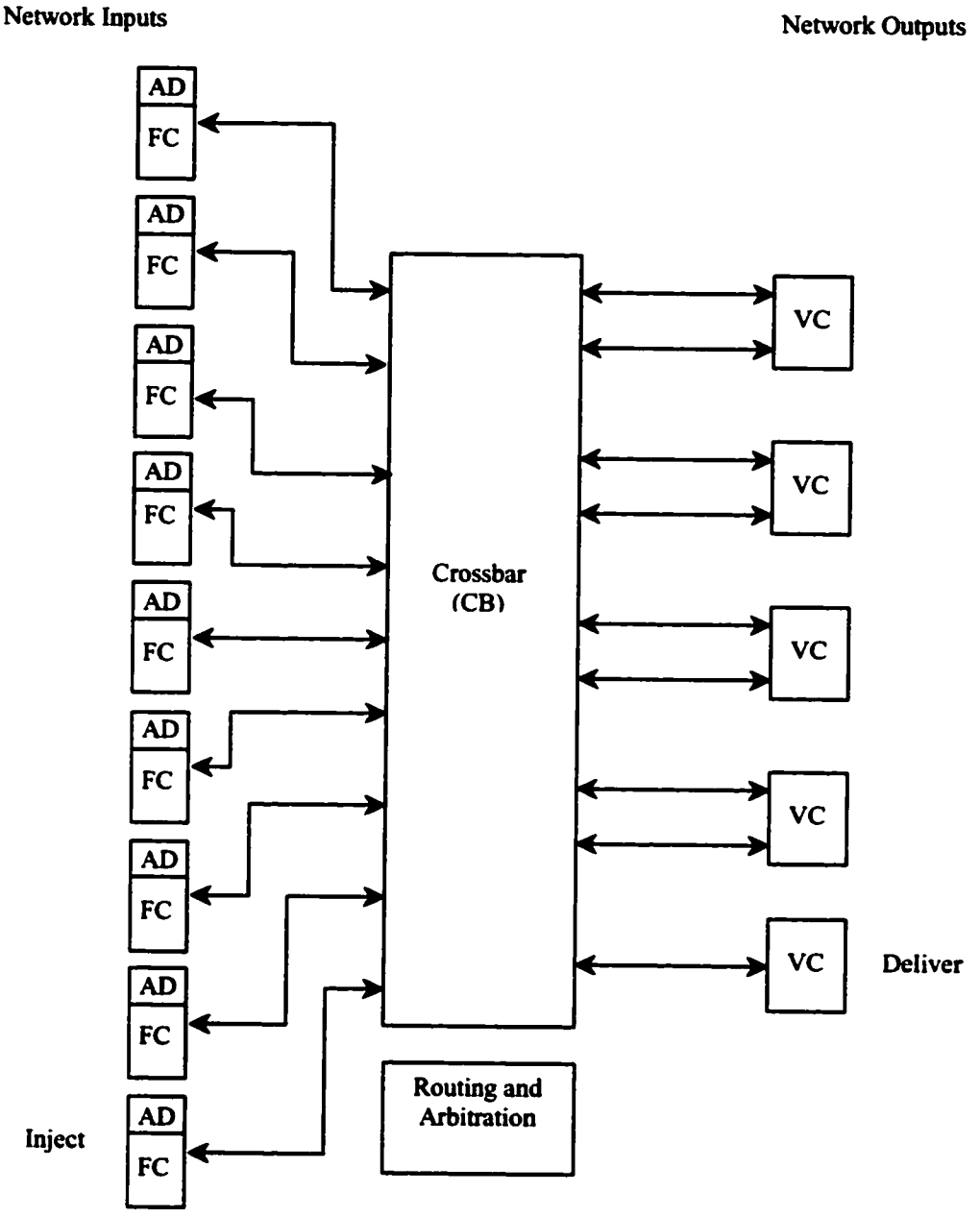


Figure 3.12. Block diagram of the *-Channel router.

The setup latency expression is:

$$T_{SC} = T_{AD} + T_{ARB} + T_{SEL} + T_{CB} + T_{VC}$$

$$T_{SC} = c_3 + c_4 + c_5 * \log_2 F + c_6 + c_7 * \log_2 F + c_0 + c_1 * \log_2 P + c_8 + c_9 + * \log_2 V$$

For a 2-dimensional network, $P = 9$, $F = 9$, and $V = 2$,

$$T_{SC} = c_3 + c_4 + c_5 * \log_2 9 + c_6 + c_7 * \log_2 9 + c_0 + c_1 * \log_2 9 + c_8 + c_9 + * \log_2 2$$

$$T_{SC} = 12.65\ ns$$

The flow control latency given by:

$$T_{fc\text{-}SC} = T_{FC} + T_{CB} + T_{VC}$$

$$T_{fc\text{-}SC} = c_2 + c_0 + c_1 * \log_2 P + c_8 + c_9 * \log_2 V$$

$$T_{fc\text{-}SC} = c_2 + c_0 + c_1 * \log_2 9 + c_8 + c_9 * \log_2 2$$

$$T_{fc\text{-}SC} = 6.5\ ns$$

The latencies incurred by the *-Channel algorithm is the worst of all the other routing algorithms evaluated using this parametric cost model. This is mainly due to the large crossbar and virtual channel requirements that make for a costly router hardware design. The advantage provided by this algorithm on the other hand is full routing adaptivity.

CHAPTER 4

PERFORMANCE EVALUATION MODEL

This chapter covers all the details and assumptions that are made throughout the performance evaluation study about the interconnection network, the router architecture, the message model, traffic patterns, and the performance metrics collected. Specifications of the network topology and configuration parameters are also reviewed, as well as the various operational parameters used by the simulator.

The simulator used to perform this evaluation along with its design structure is also discussed in more detail in what follows. The designed simulator will be used to evaluate the performance of the proposed mechanisms in comparison to a variety of relevant routing algorithms and mechanisms. The simulator will measure the important performance metrics under various traffic distributions for all algorithms evaluated.

3.1 Network Model

The router architecture assumed throughout the rest of the thesis is shown in Figure 4.1 below.

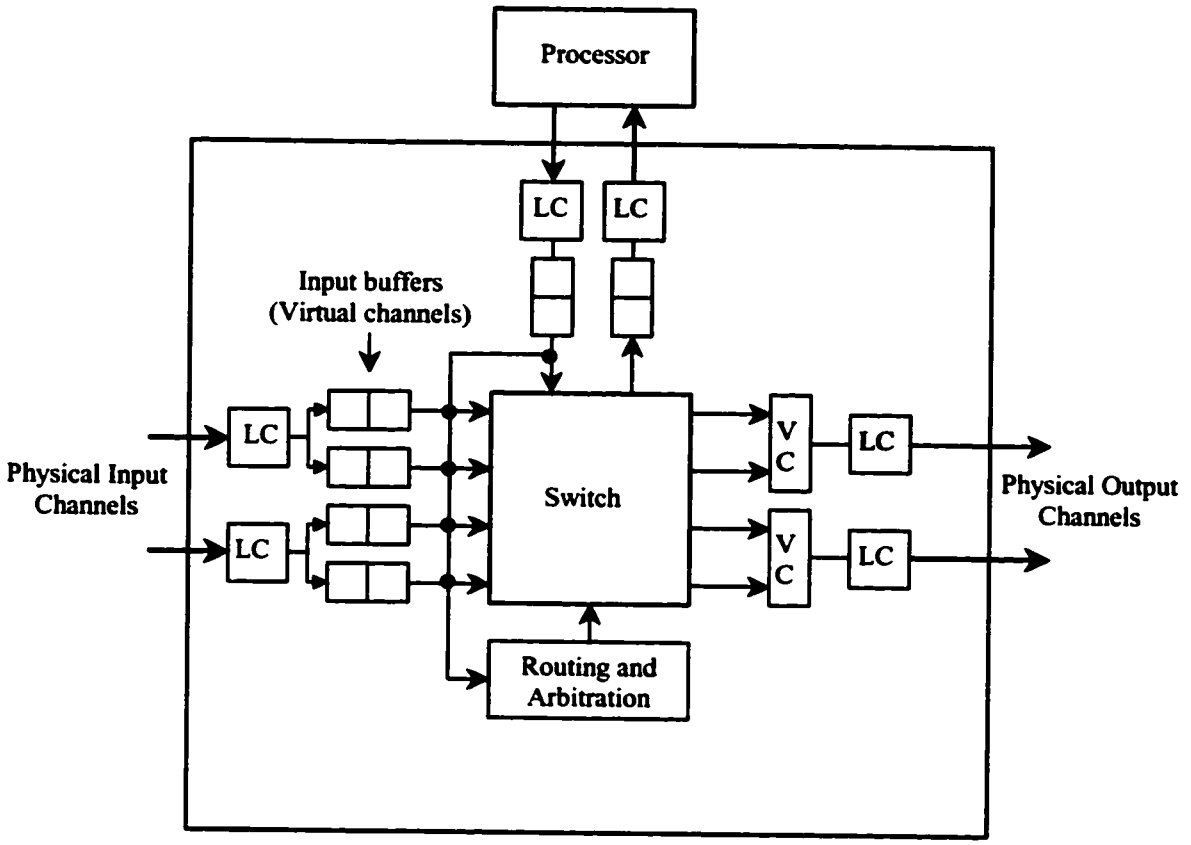


Figure 4.1. Interconnection network router architecture.

The components that make up the router architecture are:

- *Crossbar switch.* This component contains the switching fabric. It handles the movement of flits from the input buffers to the output channels. The switch has full connectivity, meaning that all input buffers are directly connected to the switch.
- *Routing and arbitration unit.* This unit implements the particular *routing function*, and *selection function* components of the routing algorithm. This unit is basically responsible for examining the headers of incoming packets, and determining the output ports that they should be routed on in order to reach their destination according to the routing function implemented. It is also responsible

for updating the headers of the packets to reflect the new offsets depending on the destination selected and on whether relative or absolute addresses are used. The simulator implements absolute addressing. The other major function of this unit is arbitration of conflicts between two or more packets that request the same output port. Whenever the requested output port for a packet is busy, the packet remains in the input buffers until that port becomes free.

- *Processor unit.* This is the local processor attached to the router to form the multicomputer node. The processor and the router are connected via one injection and one delivery channels. The simulator is configurable as to the number of injection and delivery channels used (up to the number of virtual channels implemented in the network). Only one injection and one delivery channels are assumed throughout this simulation.
- *Input buffers.* These are multi-flit FIFO buffers used to store the flits of incoming packets. This router architecture is *input buffered*. *Output buffered* models where output buffers is connected to the output ports, and *input and output buffered* models are also feasible.
- *Link Controller (LC).* The unit is responsible for implementing the flow control mechanism used by the router. The main function of this unit is to regulate the flow of flits and phits across the physical channels between adjacent nodes so that buffer overflow is prevented. This unit implements a simple Request/Acknowledge protocol for the safe transfer of flits.
- *Virtual channel controller (VC).* This unit is responsible for multiplexing the physical channel amongst all the virtual channels that share the physical medium.

The following are further assumptions made about the operation of the multicomputer and its interconnection network, as it will be used throughout the simulation studies in this thesis:

- A single injection channel between the processor and the router is used, and implies that if the processor generates a new packet while another is being injected into the network, the latter packet is queued at the tail of the injection queue.
- Injection queues are allowed to grow without bounds. Latency figures calculated include source

queuing time.

- A single delivery channel between the processor and the router is used, and means that only one flit can be delivered to the processor across this channel per cycle. If more than one packet is being delivered, delivery channel is used in a round robin fashion amongst all of the delivering packets.
- Flits passed to the delivery channel are assumed to be consumed immediately by the processor unit.
- The router of each node is connected to neighboring routers via dual uniplex channels, one for each incoming and outgoing directions, and in a full-duplex fashion.
- Each uniplex channel is associated with at least one multi-flit FIFO queue. This queue is considered to be the output queue of the sending node, and the input queue of the receiving node (input buffered architecture.)
- A physical channel is associated with a configurable number of multi-flit queues, referred to here as virtual channels, or edge buffers. The virtual channels are multiplexed over the physical channel. If more than one virtual channel can utilize the physical channel, channel bandwidth is assigned to the virtual channels in a demand-slotted round robin fashion.
- The phit size is equal to the flit size, therefore a flit can be transferred from an input queue to an output queue, and therefore across the physical channel, within one clock cycle.
- A header flit can be processed by the node's routing unit within one clock cycle.
- Only one header is processed by the routing unit at a time. If more than one header requires the routing unit, they are serviced in a round robin fashion.
- The crossbar used within the router is non-blocking. This means that it allows simultaneous connections to exist through it. Flits can be transferred from all the established crossbar connections simultaneously.

4.1.1 Network Topology and Configuration

The interconnection network topology used throughout this thesis, and to obtain all simulation results is a 256-nodes, 2-dimensional, 16x16 mesh topology network. The network size was selected for reasonable simulation times. Each physical channel is associated with three virtual channels. Each virtual channel is composed of a FIFO buffer that is 2-flits deep. Packets used are all 32-flits long, while flits are 32-bits wide. The selection function, which selects a single path from the set of all possible paths to route the packet through, will attempt to locate a free channel for the packet first. If that is not possible, it selects amongst the choices according to the *straight-first* flavor of selection. The timeout value, after which a packet is marked as deadlocked, is 10 clock cycles for all traffic patterns except the hot spot pattern, where a timeout value of 35 cycles is used.

4.2 Message Model

The following are the assumptions made about the packets in the network:

- A packet is broken down into flits. Each packet contains a header flit, data flit(s), and a tail flit.
- Header is assumed to fit in one flit, so routing can take place as soon as the header flit arrives at the input queue of a node.
- Packet generation rate is constant and the same for all nodes for a given run as determined by the load rate of that run. The calculation of the load rate is derived in 4.2.1 below.

4.2.1 Traffic Generation Rate

When running simulations with a synthetic workload, the traffic *generation rate* is used as the input parameter that determines the rate at which each node in the network generates packets. It is also known as the *applied load*, *offered traffic*, or *injection rate*. All the traffic generation rates are

normalized with respect to the maximum wire capacity of the particular network topology used. The maximum wire capacity is the injection rate at which all the physical channels in the network are utilized in each clock cycle. Since on average, each packet utilizes the channel for each unit of distance it travels, the maximum wire capacity can be calculated as the total number of wires divided by the average distance in the network. Assuming uniform traffic, the maximum wire capacity represents the maximum sustainable generation rate in flits that the network is capable of. The generation rate per node can be calculated by dividing the total generation rate by the total number of nodes in the network [36]. Following are the derivation equations for n-dimensional mesh networks with radix k:

The total number of nodes in the network is N, where:

$$N = k^n$$

The total number of channels in the network, assuming two separate uniplex channels between adjacent nodes is:

$$W = N * (2 * n) - (2 * n) * k^{n-1}$$

$$W = 256 * (2 * 2) - (2 * 2) * 16^{2-1}$$

$$W \approx k^n * (2 * n)$$

$$W = 960$$

The average distance can be calculated as follows:

$$D = n * \frac{1}{k^2} * \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} |i - j|$$

$$D = n * \left(\frac{k}{3} - \frac{1}{3k} \right)$$

$$D = 2 * \left(\frac{16}{3} - \frac{1}{3(16)} \right)$$

$$D \approx \frac{nk}{3}$$

$$D = 10.625$$

The maximum wire capacity is then:

$$\Lambda_{max} = \frac{W}{D}$$

$$\Lambda_{max} \approx 6 * k^{n-1}$$

$$\Lambda_{max} = 90.3529$$

The maximum generation rate per node becomes:

$$\lambda_{max} = \frac{\Lambda_{max}}{N}$$

$$\lambda_{max} = 0.011 \qquad \text{flits per node, or}$$

$$\lambda_{max} = 0.35294 \qquad \text{packets per node}$$

Due to network bisection limitations, maximum wire capacity utilization in mesh networks cannot be achieved. The bisection for a dual uniplex channels network can be calculated as:

$$B = 2 * k^{n-1}$$

For uniform traffic, half the traffic will cross the network bisection. The maximum achievable throughput performance can be calculated as:

$$\Phi_{max} = \frac{B}{\Lambda_{max} / 2}$$

$$\Phi_{max} = \frac{2 * k^{n-1}}{3 * k^{n-1}}$$

$$\Phi_{max} = \frac{2}{3}$$

Therefore the network can be saturated with a generation rate that is only 66/% of the maximum wire capacity. Accordingly, and to generate a better charting scale, the maximum generation rate assumed in the simulation is 66% of the maximum generation node per node, which is referred to as the saturation generation rate:

$$\lambda_{sat} = 0.011\left(\frac{2}{3}\right)$$

$$\lambda_{sat} = 0.007333 \qquad \text{packets per node}$$

This generation rate is considered to be equivalent to the maximum normalized load rate of 1.0. Any other load factor less that 1.0, such as 0.5 is accordingly a fraction of that generation rate.

A node in the network generates traffic according to a particular normalized input load rate by using the inverse of that load rate, referred to as the *injection period*. The maximum injection period is then:

$$\rho_{\max} = \frac{1}{\lambda_{sat}}$$

$$\rho_{\max} = 136$$

After generating a packet, a node waits for a random number of cycles before generating the next one. This random number of cycles is bounded between 0 and the current injection period value. The current injection period is determined by dividing the maximum injection period of the network by the

specified normalized load rate. When the load rate is set to the maximum value of 1.0, then the injection period would equal the maximum injection period. The normalized load rate is used to drive the packet generation for a particular run, and is used to plot all the results in the simulation sections.

4.2.2 Traffic Patterns

The synthetic workloads used in the performance evaluation attempt to simulate the various communication patterns that are exhibited by real parallel applications in message-passing interconnection networks. Since it is very important for a particular network and routing algorithm to perform well under different communication patterns, accordingly various traffic distribution patterns are selected to evaluate the performance characteristics of the proposed techniques. The traffic pattern or the distribution of destinations dictates the destination of the next packet generated at each node. *Uniform* and *non-uniform* traffic distributions are used in the evaluation. For non-uniform distributions, various permutations are used such as the *bit-reversal*, *dimension-reversal*, and the *hot spot* traffic patterns:

Uniform traffic. This is the most popular traffic distribution used in evaluating interconnection networks. Packet destinations are chosen uniformly amongst all the nodes in the network. More specifically the probability of a node i sending a packet to node j is the same for all i and j, where $i \neq j$.

Bit-reversal traffic. Using bit-reversal traffic, a node with binary address $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ sends a packet to a node with binary address $a_0, a_1, \ldots, a_{n-2}, a_{n-1}$. This traffic pattern causes nodes on certain rows to send packets to nodes on certain columns, causing various pockets of conflicts near the center of the network.

Dimension-reversal traffic. The dimension-reversal traffic pattern causes a node with address xy to send packets to node yx. For 2D networks this is the same as the matrix transpose traffic pattern. It has a similar effect to the bit-reversal pattern, but tends to concentrate the conflicts along the diagonal lines of the network. In our simulation, whenever the source and destination node addresses of a generated packet are the same, a packet with a random destination is injected instead, so as to allow for more

deadlocks to occur.

Hot-spot traffic. The hot spot traffic pattern used directs 5% of all the network traffic towards a single node that is randomly selected, while the remaining traffic is uniformly distributed. This pattern causes serious congestion near the hot spot node, which propagates through to other parts of the network. This causes all the routing algorithms to have early saturation points.

4.3 Performance Metrics

The two most important performance evaluation metrics for interconnection networks are the latency and throughput measures.

Latency is defined as the number of clock cycles that has elapsed from the time the packet has been generated by the processor, until the tail flit of the packet has been delivered to the destination node. This definition takes into account the injection queues, and therefore includes source queuing time. The final latency figure presented for each run is the average latency of all the packets delivered by the network.

Throughput is the maximum amount of information delivered per cycle by the network. It can also be defined as the maximum *accepted traffic* by the network. It is measured here as the maximum number of packets delivered per unit of time by the whole network. The throughput figure can be derived directly from the current load rate unless the network is or beyond its saturation point.

These two performance metrics are only collected by the simulator after the network has reached the *steady state* since they may fluctuate at the initial *transient state* of the network. The end of the transient state is detected when the flit injection rate and the flit delivery rate have stabilized. More precisely the transient period is declared over when the flit injection rate and the flit delivery rate are within 0.05% of each other for two consecutive 500-clock cycle periods. Statistics are collected for 50,000 clock cycles plus the number of cycles it took to reach the steady state. This was found to be sufficient for the simulated network size.

A steady state solution exists if the flit generation rate and the flit delivery rate are within the same range; otherwise the network will be saturated for that run. If a steady state solution exists, then a simulation run can be terminated whenever the metric measured is within 0.05% of its previous value for 10 consecutive 500-clock cycle periods. This is to save simulation time and resources.

The clock cycle duration is assumed to be constant for all routing algorithms regardless of the complexity of their respective hardware designs, and the impact that complexity has on the clock cycle.

4.4 Simulator

There are two main methods for evaluating the performance of wormhole switched interconnection networks. The first method is using analytical models and equations. The second method relies on stochastic simulation. It is very cumbersome and complex to develop accurate and general analytical models for wormhole switching due to its nature of interdependencies amongst packets, channels, and buffer resources [20, 44, 56, 57]. There are very few analytical models that can be used for the general purpose of evaluating new mechanisms; therefore performance evaluation via stochastic simulation has been the preferred method amongst researchers in this field [58].

The simulator developed as part of this thesis is a general-purpose, discrete-event, flit-level stochastic simulator for wormhole switched interconnection networks. It was developed in Java, a true object-oriented language, and it is 5000 Lines of Code (LOC). The simulator, *WormSim*, is configurable for a wide range of topologies, network sizes, routing algorithms, selection functions, arbitration policies, and various configuration parameters, such as the number of virtual channels, buffer size of each virtual channel, and the packet length.

The simulator was written in Java for various reasons. Most importantly are the object-oriented programming features of modularity, hierarchy, extensibility, reusability, and flexibility. Also the language has well-developed graphical extensions for front-end-interfaces. This will make future code developments to include a simulation visualization interface to be relatively less cumbersome. The

following figure illustrates the structure of the *WormSim* simulator and the objects that define this structure:

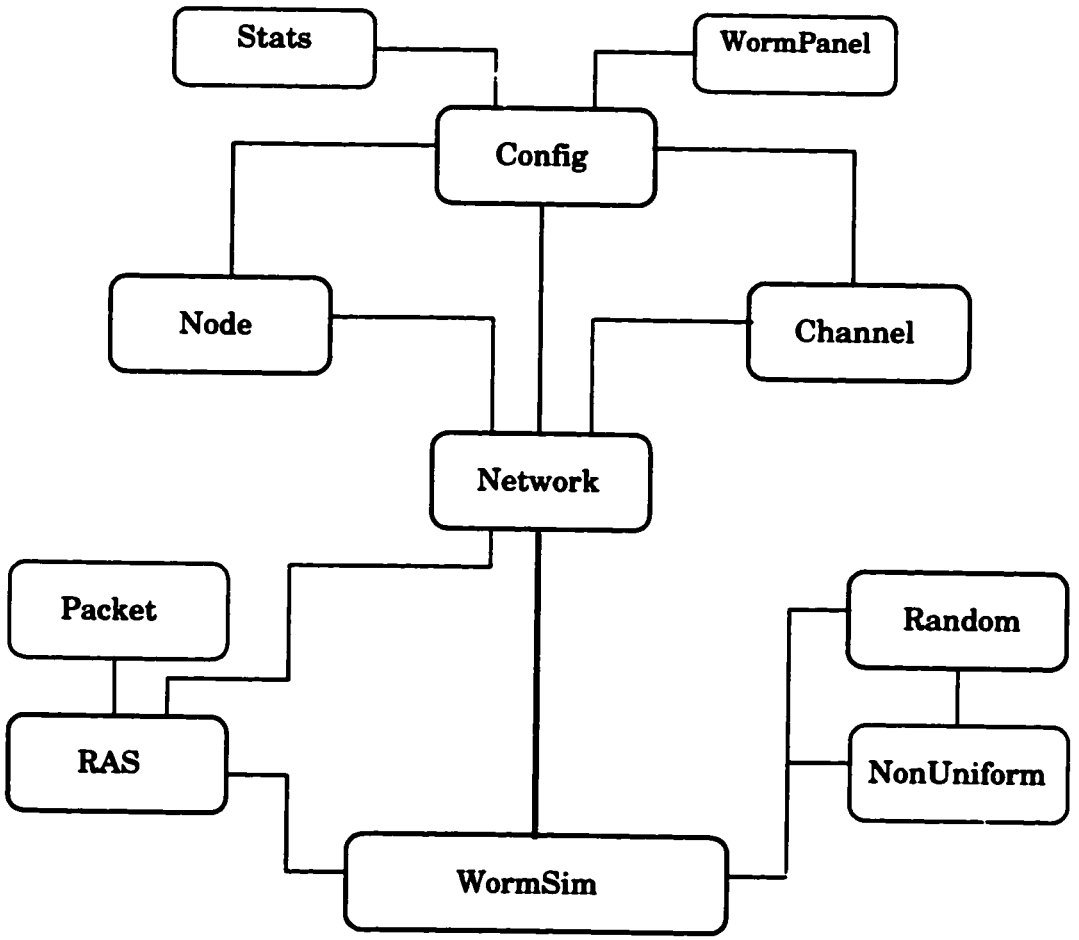


Figure 4.2. *WormSim* Simulator object structure.

Following are the general description of the above objects and their functions:

- ***Stats***. This object contains all the statistics collected for a particular run. It not only contains the essential latency and throughput metrics, but various other statistics are collected as well, such as the injection rate, delivery rate, number of packets injected, number of packets delivered, channel utilization, buffer utilization, number of deadlocks detected and recovered from (if any), and so on. All these performance metrics and statistics are saved in an external file at the end of each run.
- ***WormPanel.*** This is the object that implements the front-end graphical interface panel that shows all the statistics collected above for the current run. The statistics are updated dynamically on the panel as the metrics change.
- ***Config.*** This object has all the global definitions of the network topology and the configuration parameters. It contains the dimension n of the network, radix k, number of virtual channels used, buffer sizes, packet length, timeout values used, and so on. All other classes instantiate this object so they all can share and update the global variables and state of the network.
- ***Node.*** This object represents a node in the interconnection network. It contains all the variables and components of a node, such as the node ID, and the crossbar switch. This object has the *inject* and *deliver* methods for injection and delivery of packets in and out of the network, respectively.
- ***Channel.*** The channel object is an abstract that encapsulates the physical channel, the virtual channels multiplexed over the physical channel, and all the flow control wires and signals that connect two adjacent nodes. Each pair of node classes are connected via one channel class.
- ***Network.*** This object allocates as many nodes and channel classes, and connects them according to a particular network topology and size as specified by any k-ary n-cube class network.
- ***Packet.*** This object represents a packet. Whenever a new packet is to be injected into the network, this object is instantiated and initialized, then placed at the tail of the injection queue. This object has various parameters for a packet, such as the packet ID, header, data, and tail flit

identifications.

- ***RAS.*** This is the object that has all the routing algorithms, the various arbitration policies for incoming and outgoing packets, and the selection functions implemented. Most of the hard work is implemented within this object. This object handles the actual transfer of flits in the network, and across the switch as well.
- ***Random.*** This object implements a random number generator. More specifically it implements a Linear-Congruential Generator (LCG) of the form $X_n = a * X_{n-1} \bmod m$, where a is called the multiplier, and m is the modulus. The object implements the *getRandom* method that returns a random value between 0, and 1. The algorithm supports 100 different streams simultaneously, with a separation of 100,000 between each two streams. The random numbers generated were tested using the Chi-Square, and Kolmogrov-Smirnov (K-S) tests [59].
- ***NonUniform.*** This object extends the Random class and is responsible for generating all the remaining non-uniform traffic patterns that are used in the performance evaluation, such as the bit-reversal, and dimension reversal traffic distributions.
- ***WormSim.*** This is the main object that extends or instantiates all the other objects in the hierarchy. This object is the main routine that creates the network, generates packets according to various traffic distributions, maintains the clock cycle, and updates all collected metrics and statistics. It also determines when to terminate a particular run according to various predetermined conditions as explained in section 4.3.

There are various other objects in the simulator used for the general purpose of creating and maintaining queues and lists, such as Element, List, Queue, and QueueList. Also there are several testing and verification object modules.

CHAPTER 5

PREEMTIVE DEADLOCK REACOVERY MECHANISM

This chapter presents the proposed algorithm. It is a TFAR algorithm along with a new deadlock-recovery mechanism. It attempts to provide deadlock-recovery using minimal hardware resources, and more importantly this extra hardware is not on the critical path of routing normal packets so as not to affect the overall speed of the router. The proposed algorithm is referred to as *ZOMA* using the initials of the author and the thesis committee advisors, namely (*Zaki*, *Obaidat*, *Mulhem*, and *Al-Bassam*).

Deadlock is the main obstacle that adaptive routing algorithms in wormhole switched networks has to overcome. Adaptive algorithms should attempt to provide the maximum adaptivity possible using a minimum set of hardware resources. Since deadlocks rarely occur, all deadlock-avoidance algorithms either have high resource requirements, or do not utilize their allocated resources efficiently. True fully adaptive routing algorithms provide the highest routing flexibility. They route packets without any restrictions and are therefore susceptible to deadlock formations. These algorithms should be accompanied by an efficient deadlock-recovery mechanism that does not require expensive hardware resources, and can drain deadlocked packets from the network faster than they can degrade the performance of the network. This chapter presents a new deadlock-recovery mechanism that is preemptive in nature. The mechanism capitalizes on the wormhole switching principle of low hardware resource requirements. The performance of the proposed mechanism is on par with other progressive deadlock-recovery mechanisms; while requiring lower hardware resources that do not affect the speed of the normal routing process. The proposed mechanism creates a new category of deadlock recovery techniques that we call *preemptive* as opposed to the existing progressive and regressive categories.

5.1 Deadlock Detection Mechanism

Deadlocks can be detected in a *centralized* or *distributed* fashion. A centralized deadlock detection mechanism may not be feasible, as it requires all the nodes in the network to exchange information about their status, which cannot be implemented with efficiency being in focus. There are mainly two types of *distributed* deadlock detection mechanisms proposed in the literature, *source node* and *intermediate node* based mechanisms. *Source node* based detection mechanisms detect deadlocked packets at the source node where the packet is being injected. In this case a packet is considered to be deadlocked if a certain amount of time has passed since the packet, or the last flit of that packet was injected into the network [43]. In an *intermediate node* detection mechanism, a packet is determined to be involved in a deadlock at the intermediate nodes along the route taken by that packet, whenever it has been idle for a certain amount of time [45]. Both of these mechanisms are based on a timeout heuristic, where the packet is presumed to be deadlocked after it remains blocked for a period of time that exceeds a predetermined threshold. The main goals of a deadlock detection mechanism is to detect as much as possible true deadlocks and not false deadlocks, which are those packets merely delayed due to congestion, and to be simple enough that it can be implemented without much overhead. Efficient deadlock detection is important to the success of deadlock recovery techniques. Research in this area is still evolving in search of a better detection mechanism [60].

In this simulation we utilize the timeout-based, distributed detection heuristic as suggested in [40]. This efficient mechanism attempts to detect only one of the packets involved in a deadlock cycle, as the removal of this packet from the cycle will release enough resources to break the deadlock. Also the mechanism is not largely affected by the presence of long and short packets in the network. This case makes the selection of an appropriate timeout threshold value very difficult.

The mechanism develops the idea that blocked packets form a tree. To break the deadlock cycle only the packet at the root of the tree needs to be cleared. When a packet is blocked waiting for a resource that is occupied by a blocked packet, that packet is not the root of the tree, and therefore should

not detect a deadlock. But when a packet is blocked waiting for resources used by a non-blocked packet, which becomes blocked at a later point in time, then this packet should detect deadlock because it may be waiting on a packet that is at the root of the tree.

To implement this mechanism, each physical output channel needs an associated counter and two one-bit flags, the *Inactivity* (*I*), and the *Deadlock Threshold* (*DT*) flags. The counter is incremented at each clock cycle that the channel remains idle, and is a measure of inactivity along that channel. At least one of the virtual channels in that channel has to be occupied for the counter to be incremented. Two threshold values are used, $t1$ and $t2$. The first threshold, $t1$, is used to measure inactivity and is set to a low value (1 in this case). The second threshold, $t2$, is used to detect the deadlocked packets, and is set to 10 for all traffic patterns except the hot spot traffic, where a value of 35 clock cycles is used. The counter value is continuously compared to the threshold values. If the counter value is greater than $t1$, then the *I* flag will be set. If it is greater than $t2$, then the *DT* flag is set. Both of the counter and the flags are reset whenever a flit is transmitted across the physical channel. To determine whether a newly arrived packet is the root of the tree or not, another one-bit flag is used, called the *Generate/Propagate* (*G/P*) flag. This flag is associated with each physical input channel, and whenever it is set to *G*, then deadlocks on this particular channel can be detected, as they indicate the root of the tree.

At the first unsuccessful attempt at routing a packet, if the input physical channel of that packet has free virtual channels, then this packet cannot cause a deadlock formation, and the corresponding *G/P* flag is set to *P*. If all the virtual channels are occupied, then all the *I* flags of the requested output channels for that packet are tested:

- If one or more of the *I* flags are not set, then some packets are advancing and the *G/P* flag is set to *G*, indicating that those packets could be the root of the tree.
- Otherwise if all the *I* flags are set, then the requested resources are occupied by blocked packets. But they are not the root of the tree, and therefore the *G/P* flag is set to *P*.

During successive attempts at routing that packet, the *DT* flags associated with all the requested output channels, and the *G/P* flag associated with the input channel are monitored:

- If one or more *DT* flags are not set, then the channels may be congested but packets have been advancing across them, and therefore no deadlock is detected.
- But if all the *DT* flags are set, and the *G/P* flag is set to *G* representing a root of the tree, then the packet is marked as deadlocked, and the deadlock recovery mechanism is invoked.
- The case where all the *DT* flags are set, but the *G/P* flag is set to *P* indicates that the packet is involved in a deadlock formation, but it is not at the head of the tree of deadlocked packets, and therefore it is not flagged as deadlocked.

Whenever any of the packets waiting in a virtual channel of a physical channel are successfully routed or free their virtual channel, the corresponding *G/P* flag is set to *P*. Also whenever the *I* flag in a router is reset, all the *G/P* flags in that router are set to *G* to accommodate and detect all deadlock cases [40]. This mechanism is implemented for all the deadlock recovery routing algorithms compared in the simulation sections for consistency purposes.

5.2 ZOMA Deadlock-Recovery Mechanism

This section proposes using the True Fully Adaptive Routing algorithm and a new deadlock-recovery mechanism, ZOMA. A TFAR algorithm routes packets on all available channels and buffer resources without regard to the possibility of deadlock. Once deadlocks are detected, the proposed recovery mechanism is invoked. The deadlock detection mechanism used here is as described in section 5.1 above. The ZOMA mechanism was first published by the authors in [61].

The ZOMA deadlock-recovery scheme proposed is a new approach. Since deadlocks are rare, only as few resources as possible should be dedicated to handle this rare event. The proposed approach takes advantage of the concept behind wormhole routing. Namely the low number of edge buffer requirements per channel, which can be as low as one flit buffer, and the flow control mechanism already in place, and which carries control information in the opposite direction to that of data flow. The proposed mechanism

also creates a new category of deadlock-recovery techniques. As mentioned before, previous known categories are either progressive or regressive in nature. The proposed mechanism in neither, and belongs to a new class of deadlock-recovery techniques that is referred to here as *preemptive* deadlock-recovery. A more elaborate discussion on the details of this concept is presented in the next section.

5.3 ZOMA Hardware Requirements

The hardware required by this approach is moderate and more importantly does not increase the complexity of neither the crossbar switch, the virtual channel controller, nor does it lie on the critical path of the routing decision for normal packets. Both of these factors have been shown to increase the cost and decrease the speed of the router [34]. This will be demonstrated in section 5.5.1 below.

A central buffer is required per node. The capacity of this central buffer should be the same as that of an edge buffer. The central buffer is connected to all the input ports of the switch and can output to any output port. It has a similar architecture to the central queue used in the Chaos router [62, 63]. Two 7-bit registers per node are needed to store the input and output switch ports. A *break/reconnect* signal wire is also needed in order to preempt (break) and resume (reconnect) the routing process of a packet. This wire is very similar to the flow control wires already used for wormhole switching. Its main function is to signal the previous node to perform either a *breakMode* or a *connectMode* operation. Instead of having two wires for each mode, we just use one wire with a dedicated bit in each node. When a break signal is received by a node, it toggles this bit, so that the next time it receives it, it will actually be interpreted as a connect signal. Also and to insure sequential deadlock recovery, a synchronization token shared amongst all the nodes in the network is required, such as the one used in Disha. The router architecture for the proposed deadlock-recovery mechanism is shown in Figure 5.1 below.

This hardware is sufficient to perform sequential deadlock recovery. That is one deadlocked packet is handled at a time. Sequential deadlock recovery will be used throughout our simulation. Concurrent deadlock recovery of more than one deadlocked packet requires the replication of this hardware as to the number of deadlocked packets that could be recovered from simultaneously, but the token synchronization

hardware is not needed and is eliminated. Concurrent recovery will be considered as a future area of research in this field.

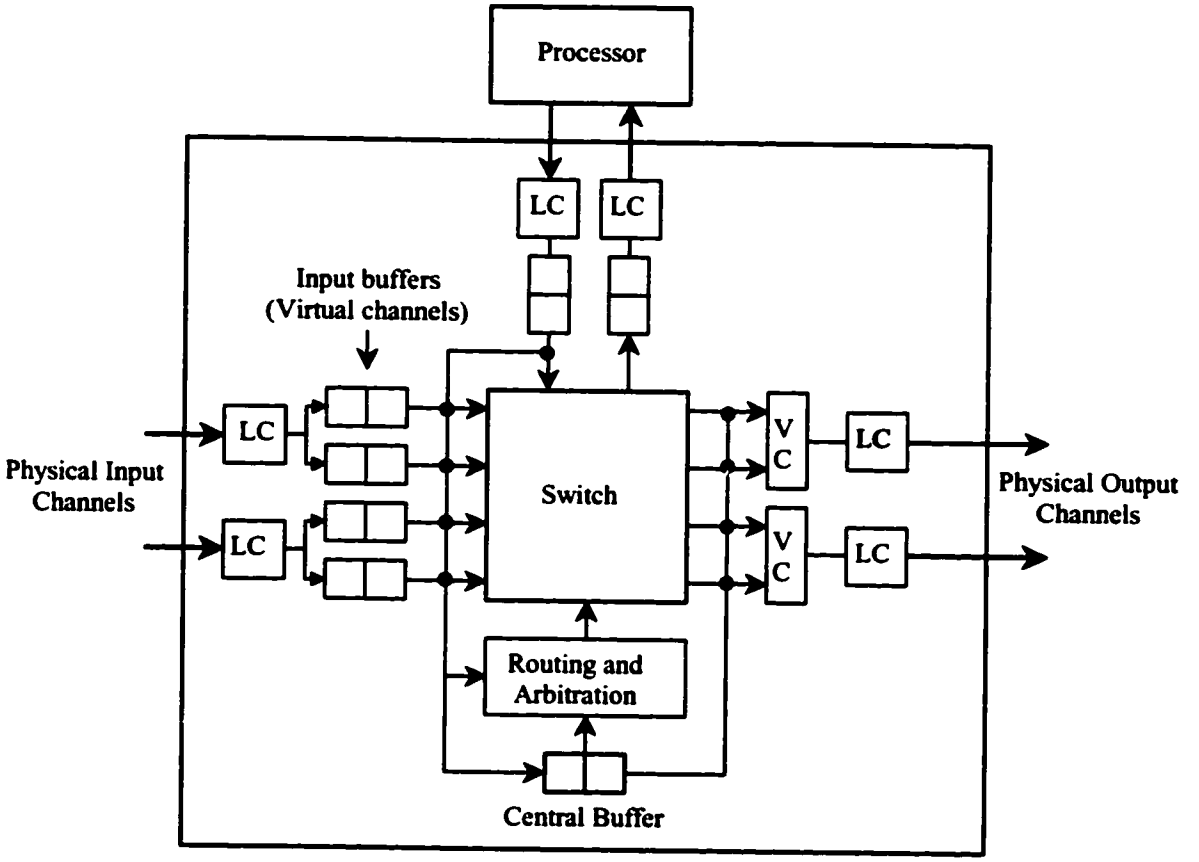


Figure 5.1. ZOMA router architecture.

5.4 Operation of the ZOMA Mechanism

The ZOMA mechanism allocates a central buffer of the same capacity as that of an edge buffer, which is usually one or two flits deep. This central buffer will be used as storage to break a deadlock cycle at the node where the header of the blocked packet resides. The flow control mechanism is also extended by one

more signal, the *break/reconnect* line. The operation of this mechanism to resolve a deadlock cycle is detailed next. Figure 5.2 illustrates a single-cycle deadlock configuration. The four routers involved in the deadlock cycle are marked as R1, R2, R3, and R4.

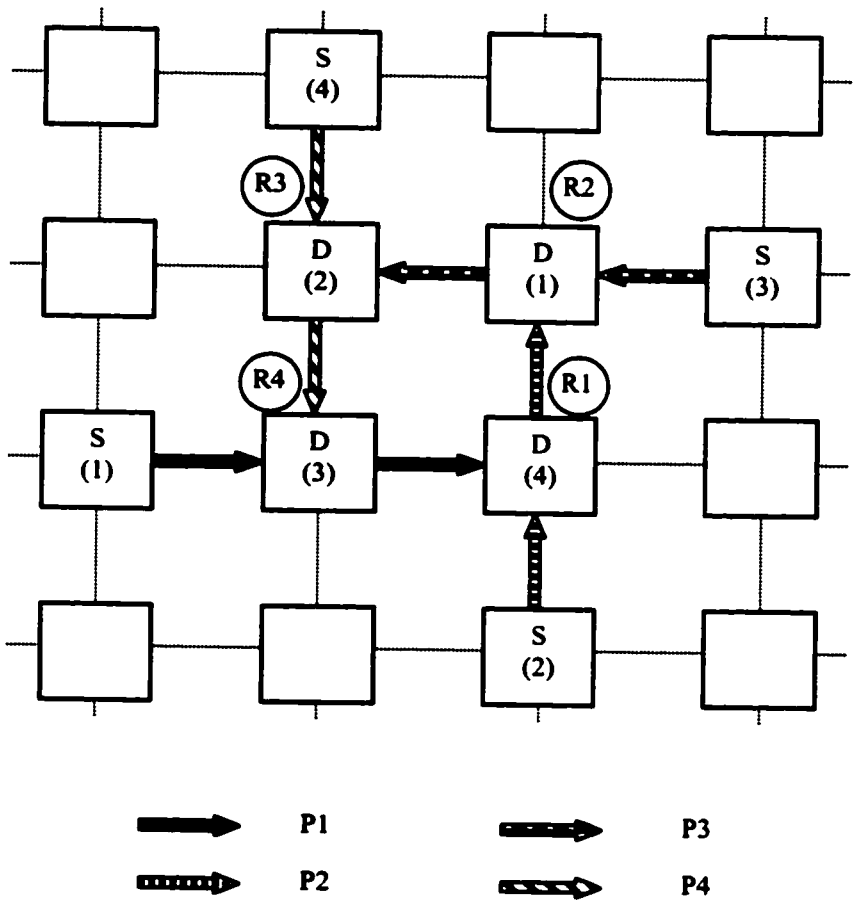


Figure 5.2. Single-cycle deadlock configuration

The above single-cycle deadlock is illustrated in more detail using the proposed router architecture in Figure 5.3 below. The four routers are marked as in the previous figure.

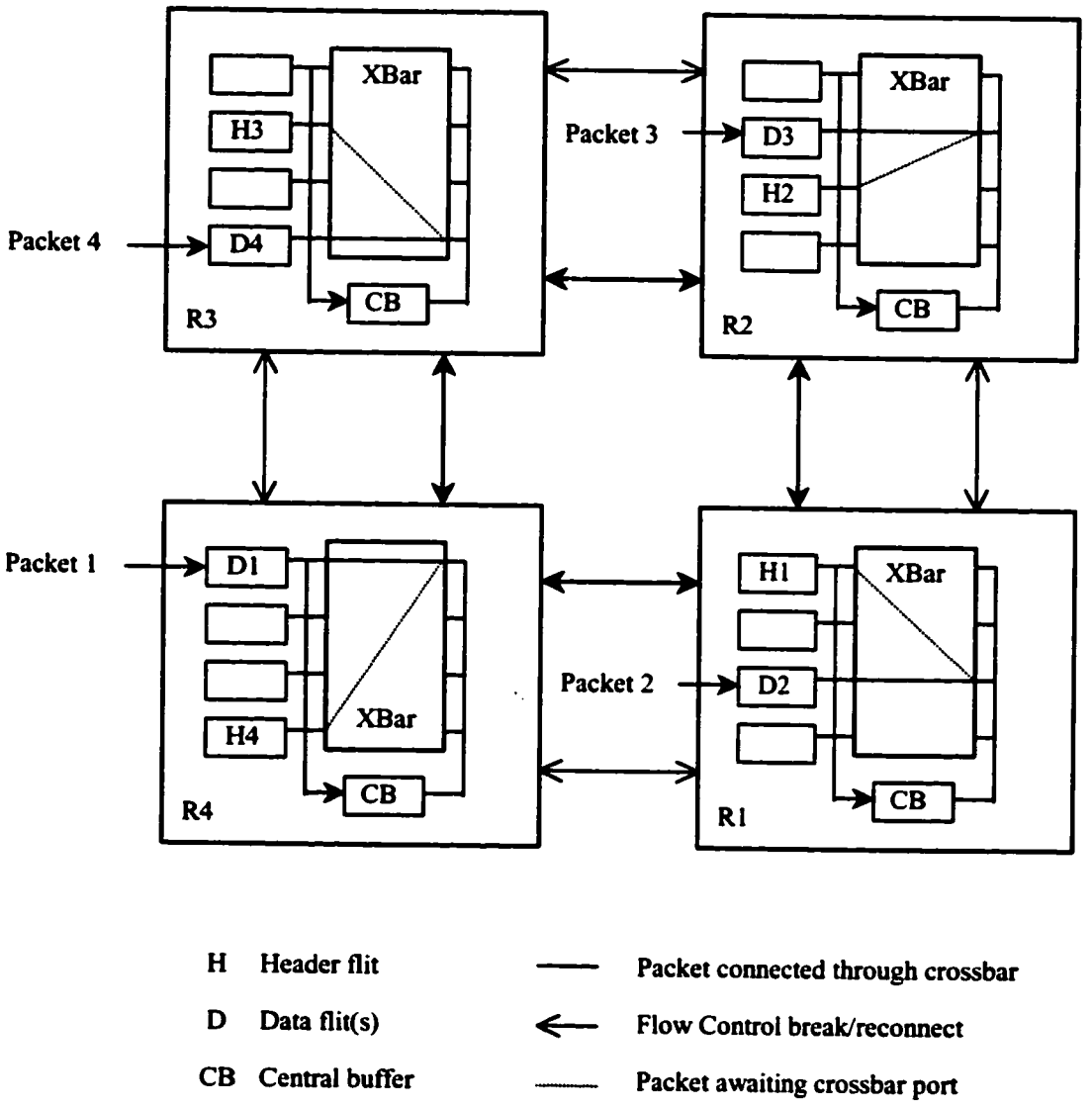


Figure 5.3. Detailed view of a single-cycle deadlock in the ZOMA router design

When a deadlock situation is detected in a node, the edge buffer that contains the header of the blocked packet is preempted and stored in the central buffer, and then cleared to receive other flit(s). Flow control in the opposite direction, using the added line, signals the previous node where the packet came

from to break and free the connection through its crossbar, to enable other packets to pass through it. This operation is referred to as a *breakMode* operation. Each node containing part of the blocked packet propagates a breakMode signal to the upstream node until the tail flit or the source node of the packet has been reached. A breakMode operation performs the following steps:

- Moves the flit(s) of the deadlocked packet from the edge buffer to the central buffer.
- Saves the input and output port numbers allocated by the packet through the crossbar of that node.
- Clears the connection made by this packet through the crossbar, so that other packets can utilize it.
- Toggles the break/reconnect bit.

Having done that, the remaining packets that are involved in the deadlock situation can now advance and will eventually free up their occupied resources. Once the node with the preempted header in its central queue has a free edge buffer, the header is rerouted again. If the routing is successful, a flow control reconnect signal is sent back to the upstream node containing the remainder of the packet in order to reestablish the connection through its crossbar, and resume the preempted operation of routing the packet. This signal, referred to as a *connectMode* operation, propagates back along the same path taken by the breakMode signal and reverses what the latter has done using the trails that were previously saved by it. A connectMode operation namely performs the following steps:

- Moves the preempted flit(s) of the deadlocked packet from the central buffer to the edge buffer.
- Reestablishes the connection through the crossbar, so as to resume the switching operation of the preempted packet. It retrieves the connection input and output ports from the saved registers.
- Clears the registers used to save the input and output port numbers.
- Toggles the break/reconnect bit.

In Figure 5.3, if Packet 1 in R1 is marked as deadlocked, then the header of the packet (H1) will be moved to the CB of that node. A breakMode signal is sent to the previous node (R4) where the data flits(s) of the packet reside (D1). The breakMode operation will move the data flit(s) to the CB, and will clear the crossbar connection established by the packet. This breakMode operation continues until the tail flit or the source node of Packet 1 has been reached. As soon as the crossbar connection is released in R4, Packet 4,

which has been blocked by Packet 1, can now advance toward its destination. This is illustrated in Figure 5.4 below.

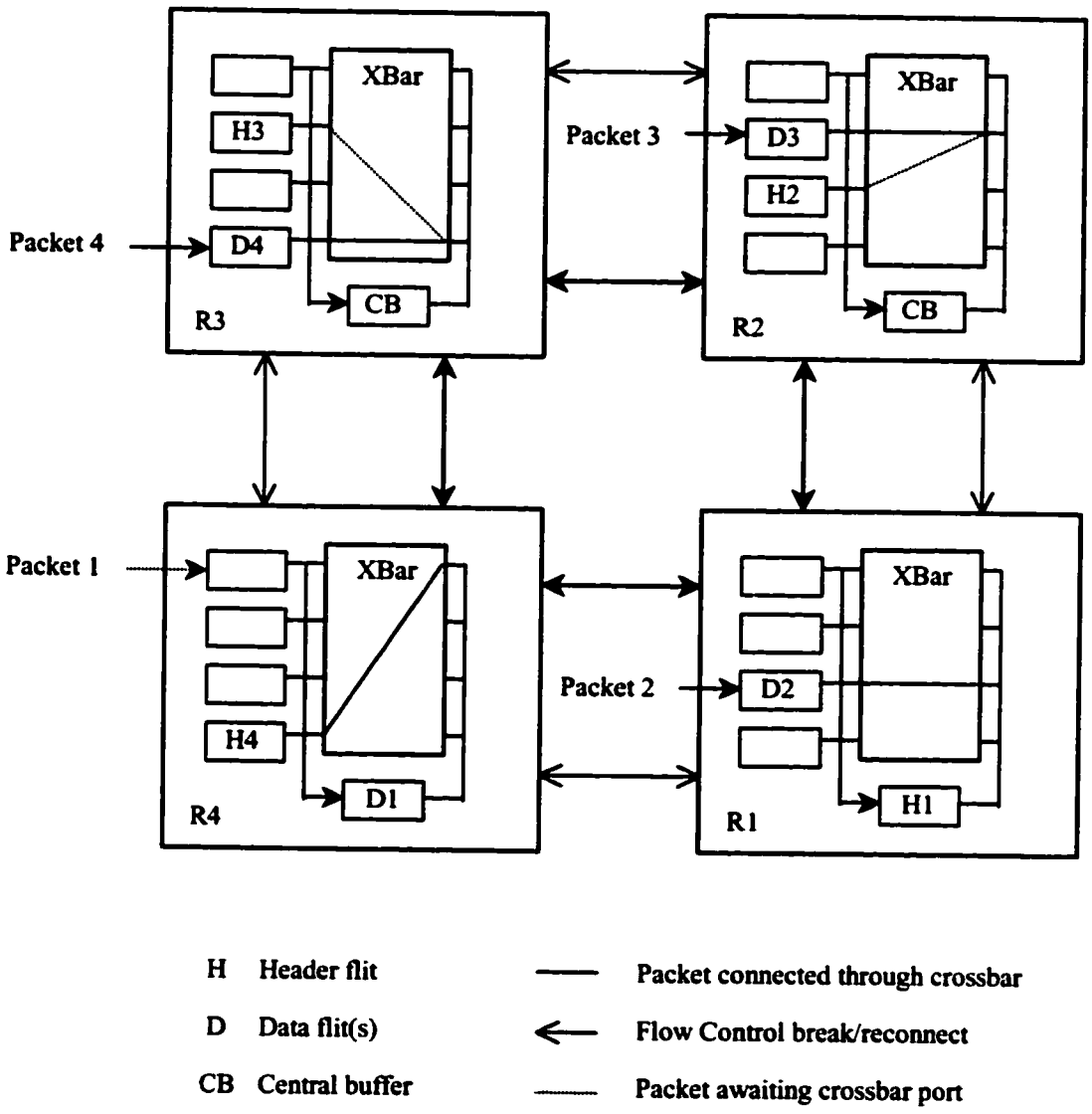


Figure 5.4. Detailed operation of the ZOMA deadlock-recovery mechanism

When Packet 4 gets routed, it will eventually release the resources it holds in R3, and R4. This breaks the deadlock cycle and allows the other packets, Packet 2 and 3, to advance to their destinations. Whenever buffer resources in R1 are freed, the preempted operation of Packet 1 is resumed.

In wormhole routing once a channel accepts the header of a packet, then it must accept all the remaining flits of that packet in a pipeline fashion. This definition does not allow for the interleaving of flits between packets on the same buffer resource. This definition is modified by this approach to state that once a channel accepts the header of a packet, then it must accept all the remaining flits of the packet eventually, and in a pipeline fashion during periods of data flow (connect mode). But here the limited interleaving of flits in order to break deadlock cycles is permitted and is implemented by the break mode operation.

5.5 Performance Evaluation

This section evaluates the performance of the ZOMA deadlock-recovery algorithm against the performance of some of the well known algorithms from the literature that represent the different categories of routing algorithms discussed earlier. The next section also evaluates the ZOMA algorithm against the similar but progressive deadlock-recovery algorithm, Disha. In section 5.5.2, the simulation results for evaluating ZOMA against Dimension-Order Routing (DOR), Planar-Adaptive Routing (PAR), Duato, and the Disha algorithms are shown and discussed.

5.5.1 Cost Model Evaluation

This section compares the cost of the additional hardware resources required by both the Disha and the ZOMA routers according to the parametric cost and speed models discussed in section 3.5. The comparison assumes a 0.8-micron CMOS gate array technology.

We first consider the *unified-crossbar* design, which consists of a single large crossbar connecting all input to output ports. It is similar to the router architecture assumed in this simulation and shown in Figure 3.12. According to the cost model in [34], the *path setup delay* of the router can be computed using:

$$T = T_{AD} + T_{ARB} + T_{SEL} + T_{CB} + T_{VC},$$

$$T = c_3 + c_4 + c_5 * \log_2 F + c_6 + c_7 * \log_2 F + c_0 + c_1 * \log_2 P + c_8 + c_9 + * \log_2 V$$

While the *flow control* latency is given by:

$$T_{fc} = T_{FC} + T_{CB} + T_{VC},$$

$$T_{fc} = c_2 + c_0 + c_1 * \log_2 P + c_8 + c_9 * \log_2 V$$

For our network topology and configuration, the proposed router has $V = 3$, $P = (2n*V) + 1$, or $P = 13$, and $F = P$. The extra input port is the injection port. The intra-router latencies are accordingly calculated as:

$$T_{ZOMA} = c_3 + c_4 + c_5 * \log_2 13 + c_6 + c_7 * \log_2 13 + c_0 + c_1 * \log_2 13 + c_8 + c_9 + * \log_2 3,$$

$$T_{ZOMA} = 13.95\ ns$$

$$T_{fc\text{-}ZOMA} = c_2 + c_0 + c_1 * \log_2 13 + c_8 + c_9 * \log_2 3,$$

$$T_{fc\text{-}ZOMA} = 7.01\ ns$$

The Disha router has $V = 3$, $P = ((2n*V)+1)+1$, or $P = 14$, and $F = P$. The extra input ports are the injection, and the disha central buffer ports. The intra-router latencies become:

$$T_{Disha} = c_3 + c_4 + c_5 * \log_2 14 + c_6 + c_7 * \log_2 14$$
$$+ c_0 + c_1 * \log_2 14 + c_8 + c_9 + * \log_2 3,$$

$$T_{Disha} = 14.14\ ns$$

$$T_{fc\text{-}Disha} = c_2 + c_0 + c_1 * \log_2 14 + c_8 + c_9 * \log_2 3,$$

$$T_{fc\text{-}Disha} = 7.08\ ns$$

Although the difference between the intra-router latencies of these two routers is small, it gives the ZOMA mechanism a slight advantage, especially that the performance results of the next section are very close and since the intra-router latencies are in effect larger for the Disha case. Also note that these values represent the delays for routing or switching a single header, or data flit respectively, and are therefore exacerbated when the entire packets routed through the network are considered.

The situation is even compounded further when we consider a modular router architecture such as the *enhanced-hierarchical router*. Modular routers will more likely be used in future multicomputers. Each subcrossbar input size can be calculated using $P = 2n + C + 1$, where n is the dimension of the network, and C the number of connect channels used between subcrossbars [64].

For our network and assuming one connect channel ($C = 1$), the ZOMA router has $P = 6$, and each subcrossbar handles one of the virtual channels ($V = 1$). The inter-router latencies are computed as:

$$T_{ZOMA} = c_3 + c_4 + c_5 * \log_2 6 + c_6 + c_7 * \log_2 6 + c_0 + c_1 * \log_2 6 + c_8 + c_9 + * \log_2 1,$$

$$T_{ZOMA} = 10.99\ ns$$

$$T_{fc\text{-}ZOMA} = c_2 + c_0 + c_1 * \log_2 6 + c_8 + c_9 * \log_2 1,$$

$$T_{fc\text{-}ZOMA} = 5.39\ ns$$

The Disha router has an extra input port at each subcrossbar for the central buffer, therefore $P = 7$ and the inter-router latencies are derived using:

$$T_{Disha} = c_3 + c_4 + c_5 * \log_2 7 + c_6 + c_7 * \log_2 7 + c_0 + c_1 * \log_2 7 + c_8 + c_9 + * \log_2 1,$$

$$T_{Disha} = 11.39\ ns$$

$$T_{fc\text{-}Disha} = c_2 + c_0 + c_1 * \log_2 7 + c_8 + c_9 * \log_2 1,$$

$$T_{fc\text{-}Disha} = 5.52\ ns$$

These values are the delay through a single sub-crossbar. More than one sub-crossbar may be traversed to get the flit to the desired virtual channel, making the data-through delay additive in nature for a single flit. This again gives more of a performance advantage for the ZOMA recovery scheme.

5.5.2 Simulation Results

This section shows all the performance results produced by the *WormSim* simulator for the various algorithms and traffic patterns evaluated. The charts show the two most important performance metrics, latency and throughput. The *Chaos Normal Form* (*CNF*) of chart display is adopted in order to have a more intuitive and clear presentation of the simulation results for both of these metrics.

The performance of a particular algorithm is determined mainly by its saturation point. *Saturation point* can be defined as follows: Latency increases as the applied load to the network increases and so does the throughput achieved. The point where the latency increases and the throughput starts to level off or decreases as a function of increasing the applied load is the saturation point of the algorithm [4].

5.5.2.1 Uniform Traffic

Simulation latency and throughput results for the uniform traffic pattern are shown in Figures 5.5, and 5.6 respectively.

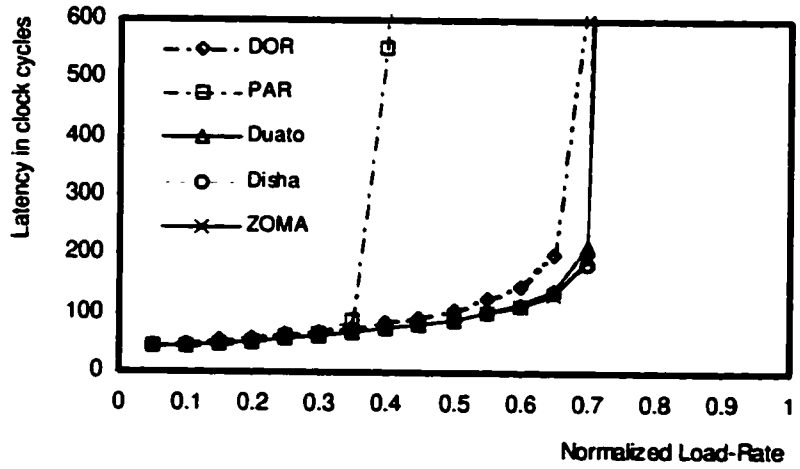


Figure 5.5. 2-dimension 16x16 mesh (Uniform traffic)

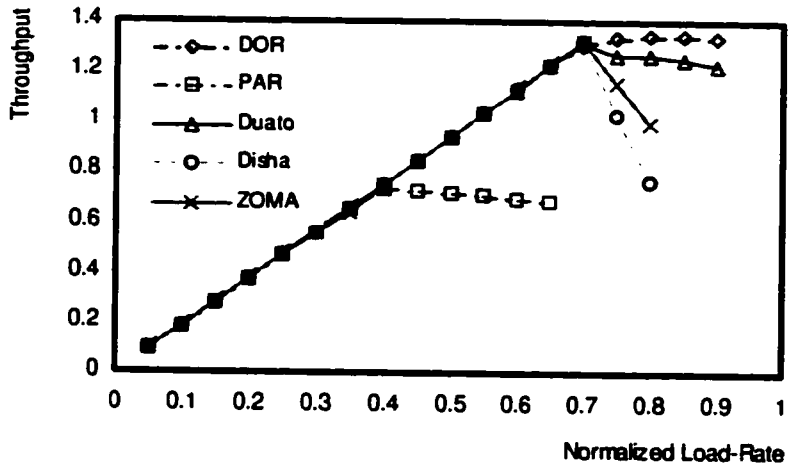


Figure 5.6. 2-dimension 16x16 mesh (Uniform traffic)

The PAR algorithm displays the worst performance; it saturates around 0.4 of the normalized wire capacity. DOR does relatively better as the traffic is already uniformly distributed, saturating around 0.68 of the normalized wire capacity. PAR demonstrates lower latencies at low load levels, but it saturates earlier as the load increases. Since traffic is already uniform, having to route the traffic in monotonic dimension order by DOR does not cause the traffic to be concentrated through any particular sets of channels, and therefore the performance enhancement. PAR on the other hand is a minimal partially adaptive router. Limited adaptivity does not provide any performance enhancements for evenly distributed traffic. Also the mesh is not a regular network topology, like the torus. Therefore adaptive algorithms generally direct more traffic toward the center of the mesh, while leaving the channels near the borders lightly utilized.

PAR reaches early saturation, as even the limited adaptivity tends to direct most of the traffic toward the center of the mesh, which creates congestion and therefore lower performance. DOR does not suffer from this problem, as routing uniform traffic in monotonic dimension order does not lead to congestion in any part of the mesh. Moreover, using PAR the virtual channels of one of the dimensions

in the mesh have to be partitioned and used separately by the increasing and decreasing packets in order to avoid deadlock. This tends to limit the available virtual channels in that dimension.

Duato Algorithm, being a fully adaptive router displays comparable performance to that of Disha and the ZOMA algorithms. Because of the mesh network, Duato only requires one virtual channel to be dedicated as the escape channel; this frees two virtual channels for adaptive routing. As the traffic is already uniform, the overall performance resembles that of the TFAR algorithms. But this behavior is not consistent for all topologies. Duato for example requires two virtual channels as escape paths in torus networks. This leaves only one adaptive virtual channel, and therefore the performance will be considerably lower than the TFAR algorithms presented here. This has been demonstrated by a number of studies [45, 46].

The performance of Disha and the ZOMA algorithms are almost identical. They both saturate at 0.7 of the normalized wire capacity. As we mentioned in section 5.5.1, this gives an advantage to the ZOMA scheme as it has lower path setup and flow control latencies. Deadlocks occur in both algorithms around the saturation point, but it is obvious that the ZOMA deadlock recovery mechanism can match that of Disha. This is in spite of the fact that the hardware resources utilized by Disha can be used to progress deadlocked packets. The resources used by the ZOMA scheme however, are not progressive in nature; they are merely used to preempt the deadlocked packet to a storage that is distributed throughout the network. This suggests that the Disha extra lane can actually be utilized for routing normal packets, which will result in achieving higher throughputs in the presence of an efficient deadlock recovery mechanism, such as ZOMA.

The throughput of both of the two TFAR algorithms demonstrate severe performance degradation after the network reaches its saturation point as seen in Figure 5.6. This performance degradation is typical of all algorithms that allow cyclic dependencies while routing, and in varying degrees according to the cyclic freedom allowed. TFAR allows the most freedom to route cyclically in comparison to all other routing algorithms, and therefore the severe performance degradation beyond the saturation point. More so, deadlocks start to form around and beyond the saturation point. Deadlocks, especially if not resolved quickly, will exacerbate the channel congestion problem, and therefore the performance

degradation. The ZOMA algorithm demonstrates less severe performance degradation than Disha, which supports the intuition that the ZOMA deadlock recovery mechanism by preempting the deadlocked packet acts as a faster draining mechanism than Disha, and allowing the deadlock cycle to be resolved quicker.

The throughput of DOR and PAR does not suffer from that performance degradation and the network remains relatively stable beyond the saturation point. This is precisely for the reason mentioned above, as those two algorithms do not allow cyclic dependencies to form amongst the packets in the network. DOR completely prevents them, while PAR disallows them on the same set of virtual channels. Duato is an interesting case, as the performance displayed beyond saturation is somewhere in between the cyclic and the non-cyclic algorithms. Although Duato is a fully adaptive algorithm, it does isolate a set of virtual channels that do not allow cyclic dependencies, namely the escape channels. Those escape channels route packets in a dimension-order fashion, and can always be utilized in order to break any cycle. The performance behavior of Duato is therefore in accordance with that notion. This is why it is the honest opinion of this author that the Duato algorithm should not be classified in the category of the fully adaptive routers, but rather as a maximally adaptive routing algorithm.

5.5.2.2 **Non-Uniform Traffic**

Three traffic patterns are evaluated in this section, *bit-reversal*, *dimension-reversal*, and the *hot spot* traffic patterns. These traffic distributions more closely resemble the communication patterns exhibited by real parallel application in message-passing interconnection networks.

Bit-reversal traffic. Using bit-reversal traffic, a node with binary address $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ sends a packet to a node with binary address $a_0, a_1, \ldots, a_{n-2}, a_{n-1}$. This traffic pattern causes nodes on certain rows to send packets to nodes on certain columns, causing various pockets of congestion near the center of the mesh network. The simulation results are shown in Figures 5.7, and 5.8 below.

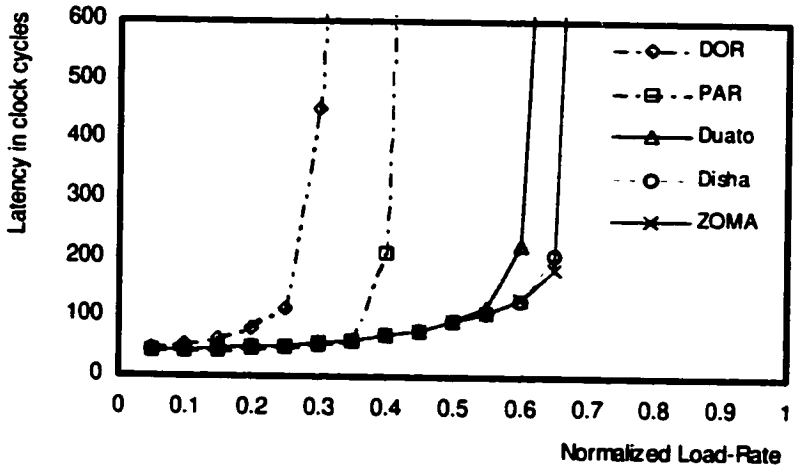


Figure 5.7. 2-dimension 16x16 mesh (Bit reversal traffic)

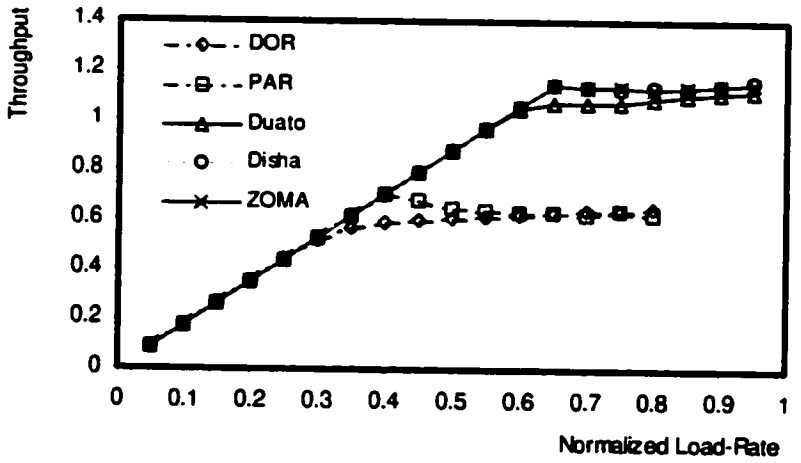


Figure 5.8. 2-dimension 16x16 mesh (Bit reversal traffic)

The uneven traffic distribution causes DOR to saturate early at around 0.3 of wire capacity, while adaptive algorithms generally do a better job in dissipating the congestion. Traffic in this pattern is concentrated over a set of channels near the center of the mesh, DOR being a deterministic router, does not attempt to avoid these channels, which causes them to be overloaded and therefore the early saturation. PAR shows better performance than DOR in this traffic pattern with a saturation point of 0.4 due to the limited adaptivity provided. Duato displays better performance than PAR saturating at 0.6 due to its higher

level of adaptivity. But Disha and ZOMA outperform the other algorithms as the true full adaptivity provided can more effectively route around congested areas than all the other algorithms evaluated. Disha and the ZOMA algorithms have almost identical performance with a 0.65 saturation point. This is a 117%, 63%, and 8% performance improvement over DOR, PAR, and Duato algorithms respectively.

Figure 5.8 shows the throughput achieved by all algorithms. The most interesting feature here is that none of the algorithms demonstrate severe performance degradation beyond the saturation point. This is due to the fact that the bit-reversal traffic pattern does not lend itself to forming cyclic dependencies while routing, as the traffic is concentrated along straight lines between certain rows and columns of the mesh. The performance of most routing algorithms demonstrate a stable network performance beyond saturation, with a moderate increase in throughput as the offered load to the network increases. The exception seems to be the PAR algorithm, where the performance shows slight degradation beyond saturation. This is due to the fact that the PAR algorithm tends to direct more traffic toward the center of the mesh because of its limited adaptivity. Added to this is the fact that the bit-reversal traffic pattern tends to concentrate the traffic near the center of the mesh, the result becomes compounded higher congestion near the center of mesh that PAR cannot effectively deal with, and therefore the slight performance degradation. Again the ZOMA and Disha algorithms demonstrate almost identical performance characteristics.

Dimension-reversal traffic. The dimension-reversal traffic pattern causes a node with address *xy* to send packets to node *yx*. For 2D networks this is the same as the *matrix transpose* traffic pattern. It has a similar effect as the bit-reversal pattern, but tends to concentrate the conflicts along the diagonal lines rather than the center of the mesh network. In our simulation, whenever the source and destination node addresses end up being the same, a packet with a random destination is injected instead, so as to allow for more deadlocks to occur. Figures 5.9, and 5.10 below show the performance metrics for all he routing algorithms in this traffic pattern.

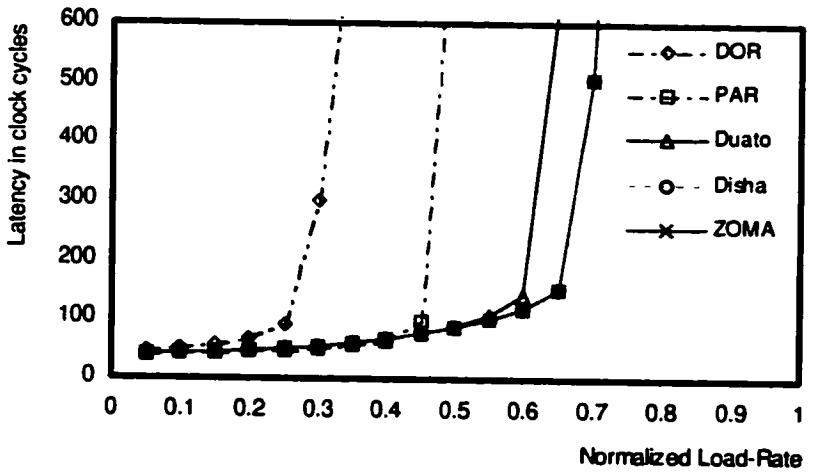


Figure 5.9. 2-dimension 16x16 mesh (Dimension reversal traffic)

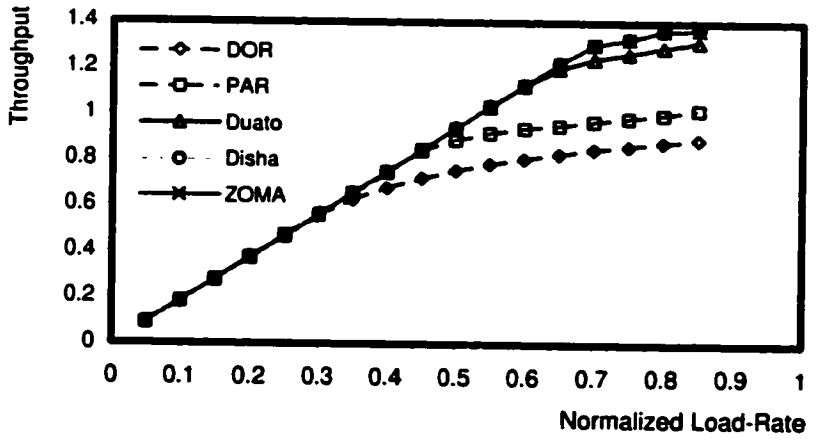


Figure 5.10. 2-dimension 16x16 mesh (Dimension reversal traffic)

Again, and as in the bit-reversal traffic pattern, the non-uniform distribution of traffic causes DOR to have an early saturation point. It saturates around 0.3 of normalized load rate. As a deterministic router it does not attempt to spread the traffic away from the congested channels. PAR improves on the performance of DOR as the partial adaptivity attempts to route around the congested areas. Albeit in a limited fashion. PAR saturates around 0.45 of normalized load rate. The Duato algorithm improves further on the performance of PAR in accordance with the higher degree of adaptivity it provides. It

saturates around 0.65 of normalized load rate. The performance of both the ZOMA and the Disha algorithms are similar, saturating around 0.7 of the normalized load rate. They outperform all the other algorithms as they provide the most adaptivity possible, which allows them to direct traffic away from congested areas, and therefore utilize more channel resources.

Figure 5.10 shows the throughput achieved by all routing algorithms. Using this traffic pattern, all the algorithms do not demonstrate any performance degradation beyond the saturation point. This traffic pattern, much like the bit-reversal, does not cause any cyclic dependencies in the network. More so, all the congestion is concentrated along the diagonals of the network, so the center of the mesh escapes the relatively higher congestion induced by the bit-reversal pattern. This is why all the algorithms manage to maintain the stability of the network beyond the saturation point. Again the performance of both the ZOMA and the Disha algorithms are almost identical. They outperform all the other routing algorithms. Their performance demonstrates a 133%, 56%, and 8% improvement over the DOR, PAR, and Duato algorithms respectively.

Hot-spot traffic. The hot spot traffic pattern used directs 5% of all network traffic toward a single node that is randomly selected, while the remaining traffic is uniformly distributed. This pattern causes a serious congestion near the hot spot node, which propagates through to other parts of the network. This causes all the routing algorithms to have early saturation points. This traffic pattern also causes serious cyclic dependencies to form around the congested areas. Figure 5.11, and 5.12 show the performance metrics for all the algorithms using this traffic pattern.

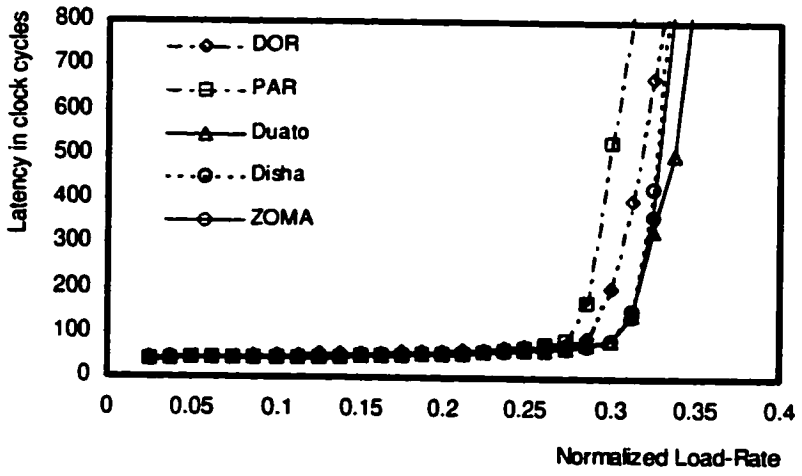


Figure 5.11 2-dimension 16x16 mesh (Hot spot traffic)

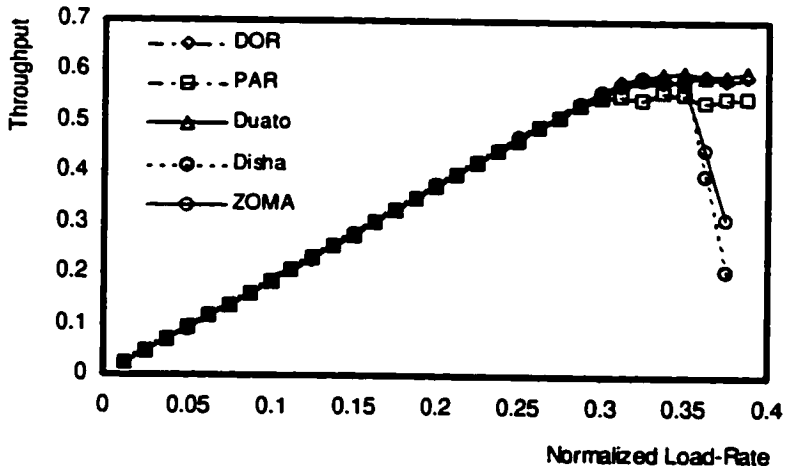


Figure 5.12. 2-dimension 16x16 mesh (Hot spot traffic)

Examining the two figures, we notice that PAR saturates at 0.3 while DOR saturates at 0.325 of the normalized load rate. DOR has a performance advantage here due to the fact that in this pattern more traffic is concentrated through particular channels. DOR has all the virtual channels available to it for mutiplexing over these channels, while PAR has restrictions on using its virtual channels, and hence the

lower saturation point. Also most of the traffic in this pattern is uniformly distributed, which gives an advantage to the DOR algorithm as explained in the uniform traffic pattern case. Duato algorithm shows a slight performance improvement over Disha and the ZOMA algorithms, as it saturates around 0.35 of the normalized load rate. It begins to saturate around the same point, but as saturation advances, the number of deadlocked packets that the TFAR algorithms have to recover from start to increase. This leads to larger latencies for the TFAR algorithms, which consequently causes them to have a somewhat steeper saturation curve. Also the Duato algorithm allocates the escape channels, which can always be used in the event of blockage. These channels do not allow cyclic dependencies, which allows the Duato algorithm to have a slightly higher saturation point. As mentioned before, the Duato algorithm does not demonstrate consistent performance for all network topologies. It requires two virtual escape channels in torus networks, and therefore performs lower than the TFAR algorithms presented here. The ZOMA and Disha algorithms both saturate roughly at around 0.3375 of the normalized load rate.

The throughput of the algorithms is shown in Figure 5.12. PAR demonstrates low throughput, as it cannot deal effectively with the congestion produced by this traffic pattern. The limited adaptivity it provides is not capable of dissipating the congestion as explained earlier. DOR and Duato algorithms demonstrate higher throughput performance as they restrict cyclic dependencies. DOR does not allow cyclic routing, while Duato provides the escape channels, which are acyclic. PAR, DOR, and Duato demonstrate relatively stable network behavior beyond the saturation point as they all restrict cyclic routing in one form or another. Both of the TFAR algorithms, Disha and ZOMA display severe performance degradation beyond the saturation point because they allow unrestricted cyclic routing, which is in accordance with observations made earlier in the uniform traffic pattern. From the figure we can also observe that the ZOMA algorithm has a less severe performance degradation behavior. This is consistent as well with previous observations made in the uniform traffic pattern and are investigated further next.

In this traffic pattern and as just noted that the ZOMA scheme demonstrates a slightly higher and less severe saturation behavior than Disha. This is due to the fact that the number of deadlocks detected by the ZOMA algorithm is much lower than those detected by the Disha algorithm. Figure 5.13

illustrates this by plotting the number of deadlocks detected normalized to the total number of packets delivered by the network as a function of the normalized load rate.

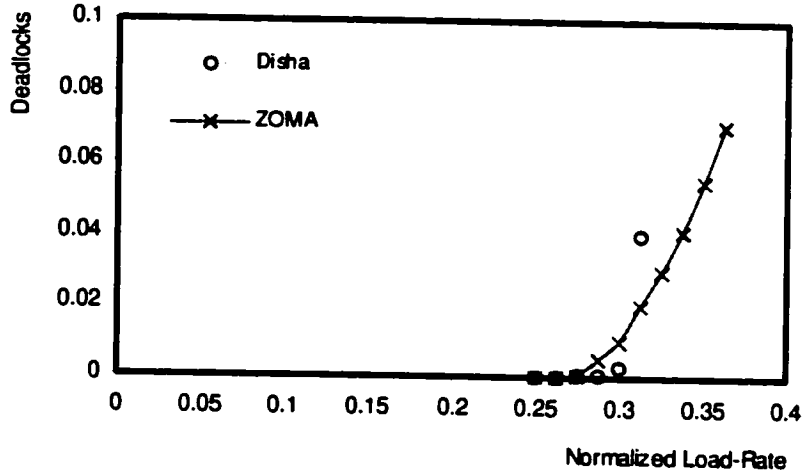


Figure 5.13. Frequency of deadlocks in hot spot traffic

From the figure, we can observe that prior to the saturation point Disha detects a lower number of deadlocks than the ZOMA algorithm. But near and beyond the saturation point the number of deadlocks detected by Disha demonstrate exponential growth, while the ZOMA mechanism maintains a linear nature. This important observation is generally true for all traffic patterns, but is more obvious in this case. The reason for this behavior is that in the ZOMA case, deadlocked packets are completely removed from the network quickly so other packets can advance. Disha on the other hand, progressively advances the deadlocked packet to its destination over a slower lane, causing more congestion to form behind it, and therefore more deadlocks. The ZOMA mechanism however, does not have a clear performance advantage because the latencies of the deadlocked packets are larger due to the fact that they wait for other packets to clear the buffer resources before they can be rerouted again. In this traffic pattern, the deadlocked packets may wait for longer periods before finding a free virtual channel to reroute the packet on. The congestion presented by this traffic pattern can be alleviated by

either increasing the number of delivery channels at each node, or by allowing misrouting as a mean of routing around the congestion points [45].

This chapter presented ZOMA, a new deadlock-recovery mechanism for true fully adaptive routing algorithms in wormhole switched interconnection networks. The ZOMA mechanism is efficient as it uses moderate hardware resources. Most importantly this hardware is not on the critical path of the routing process, and therefore does not contribute to the delay of routing normal packets as evident by the cost model calculations. The mechanism takes advantage of the concept of wormhole routing of low edge buffer requirements and the flow control mechanism, which sends control signals to regulate the flow of flits in the network. Simulation results show that the ZOMA mechanism along with the TFAR algorithm outperforms most routing algorithms and matches the performance of other more expensive progressive deadlock-recovery techniques such as Disha. Although the Duato algorithm demonstrated favorable performance in comparison to the proposed TFAR algorithm in one of the traffic patterns, it is neither consistent nor applicable to all network topologies and configurations. Duato displays lower performance in torus networks, as it requires two virtual channels to be used as escape channels. Also the Duato algorithm cannot be used in network configurations that do not meet its virtual channel requirements. TFAR algorithms are more consistent and are generally applicable to a wider class of network topologies and configuration, even those that do not support virtual channels at all.

Results show that the ZOMA algorithm causes fewer deadlocks to form by quickly removing them from the network as was demonstrated in the hot spot traffic case. The ZOMA mechanism defines a new category of deadlock recovery techniques that is preemptive in nature as opposed to the existing progressive and regressive categories. Also the new scheme is even more efficient when used with modular router designs such as the enhanced-hierarchical router, which will most likely be used in future interconnection networks. Disha requires an additional port in each subcrossbar, which will make it more expensive as demonstrated by the cost model. Also when using a router model that contains input and output buffers, there is no extra overhead incurred by the new approach. While Disha suffers from this problem, as output buffers will have to be cleared, and stored in additional hardware during

the progressive deadlock recovery phase. This extra overhead was not modeled in this thesis as input buffering is assumed throughout the simulation effort. The ZOMA mechanism can also allow for smarter misrouting techniques to be used by the nonminimal routing algorithm to route around congested areas. For example using the new mechanism it is possible to allow only those packets that have been preempted to be eligible for misrouting.

CHAPTER 6

INJECTION LIMITATION MECHANISM

As networks reach their saturation point, their performance begins to degrade. In the case of true fully adaptive routing algorithms, the network performance is severely degraded when the network is beyond its saturation point as illustrated by the simulation results of the previous chapter. The performance degradation exhibited by true fully adaptive routing algorithms results in unstable network behavior beyond the saturation point. Also as cyclic, and therefore deadlock formations start to increase near and beyond the saturation, the performance degradation problem is further compounded. These networks can benefit greatly by regulating packet injection into the network as it approaches its saturation point. These techniques are known as injection limitation or throttle mechanisms.

There are two main objectives of injection limitation techniques. First is the elimination of severe performance degradation at high load rates, which allows the network to maintain its stability beyond the saturation point. Second, these techniques are complimentary to deadlock-recovery algorithms. By reducing the frequency of deadlocks near and beyond the saturation point, efficient, low-cost deadlock-recovery mechanisms become more viable. Both of these objectives are well applicable to true fully adaptive routing algorithms, as these algorithms exhibit and suffer from both of these problems.

In this chapter a new injection limitation mechanism is proposed. The new mechanism complements the proposed true fully adaptive routing algorithm with an efficient deadlock-recovery mechanism proposed in the previous chapter, and named ZOMA.

6.1 Related Work

Injection limitation techniques were relatively recently developed to be used along routing algorithms in wormhole switched networks. In what follows, we attempt to survey and shed some light on some of the efforts found in the literature leading into the most recent work in this area.

In [65], an injection limitation technique was used along Deflection routing. Using deflection routing flow control, whenever the requested output path by a packet at an intermediate node is busy, the packet is deflected through any other available output path or channel. The main drawbacks of deflection routing are the out-of-order delivery of packets, and the unbounded transfer delays encountered due to the continuous deflection of packets. The paper attempted to minimize the effects of the unbounded delays by using an injection limitation mechanism referred to as *Input Rate Control*. The injection limitation technique works as follows: the header of each packet carries two extra fields, the number of hops that the packet has already traversed, and the number of deflections undertaken by the packet. Using this information, each node computes the ratio between these two values, which is referred to as the *Deflection Probability*. If this computed value is greater than a predetermined threshold value, then the local node is prevented from injecting packets into the network, otherwise the mechanism is not active and the node can inject packets. In this paper, the computed deflection probability value was considered to be an indicator of the level of traffic rates present in the network, and when this value increases beyond the threshold then the amount of traffic in the network is high enough to cause unbounded delays, and therefore should be limited.

A basic injection limitation mechanism was used in [66]. The paper proposed using a fully adaptive routing algorithm based on the Duato model [37, 39]. Virtual channels are divided into two classes, deterministic and adaptive channels. Injection of new packets is only allowed on a subset of the adaptive channels, which are referred to as the *source throttling* lanes. This mechanism introduces a crude injection limitation to the packet injection rate. This tends to alleviate the performance degradation problem at high network load rates, albeit in a basic or crude fashion.

Most recent injection limitation mechanisms proposed in the literature are based on controlling the number of utilized virtual channels within a node. Injection of new packets is only permitted when the number of occupied virtual channels in a particular node is less than a predetermined threshold value [13, 67, 68]. The value of the threshold is usually determined empirically via simulation just as the network enters the saturation point. By attempting to control the maximum allowed number of occupied virtual channels throughout the network, it is generally feasible to alleviate the performance degradation exhibited beyond the saturation point.

Injection limitation mechanisms can generally be classified in different categories depending mainly on how dynamic and adaptive the mechanisms are. Earlier injection limitation mechanisms were mainly *static* in nature. The threshold value must be modified for various traffic patterns and network conditions, which makes selecting the optimal value for this threshold quite difficult. If the threshold value imposes maximum restrictions, then there may be a performance penalty of higher latencies at normal network loads. While if there are less restrictions on the threshold value, then the mechanism may not be able to achieve its goal of preventing the performance degradation beyond the saturation point of the network [67, 68]. To overcome the shortcomings of these injection limitation mechanisms in relation to requiring different settings for different traffic distributions, some mechanisms update the threshold value once, after guessing the traffic distribution being used to drive the network. The threshold value is computed using a linear function. These mechanisms are accordingly referred to as *semi-dynamic* injection limitation mechanisms [69]. When the injection limitation mechanism attempts to be dynamic and adaptive to current network load, network conditions, and traffic distributions then the mechanism is referred to as a *dynamic* injection limitation mechanism [70]. These mechanisms attempt to dynamically determine the appropriate threshold value depending on the current network and traffic conditions. A selected dynamic mechanism from the recent literature is considered in more detail next, and is also used for comparison in the simulation sections.

6.1.1 Dynamically Reduced Message Injection Limitation (DRIL) Mechanism

The *DRIL* (*Dynamically Reduced message Injection Limitation*) mechanism is a dynamic injection limitation mechanism. It is based on approximating network traffic by counting the number of busy virtual channels used within the nodes of the network, and dynamically adapting to this calculated network load. The DRIL mechanism has been shown to outperform the other mechanisms that are based on counting the number of busy virtual channels, and will therefore be used for comparison and evaluation purposes. The mechanism works simply by preventing the injection of new packets into the network if the number of busy virtual channels surpasses a predetermined threshold value. Since the optimal threshold value is dependent on several variables that range from network topology to packet length, it is determined empirically via simulation. The number of busy virtual channels is calculated just before the network reaches its saturation point. This value is then used as the basis for the initial threshold value of the mechanism. The threshold value is then dynamically updated as a function of network load. Each node dynamically determines its own optimal value of the number of busy virtual channels by observing their count as the network enters its saturation point. Being near the saturation point is estimated by the number of packets that are queued for injection in a particular node. When the number of packets in local injection queues exceeds an empirically predetermined value, then a count of the number of busy virtual channels is performed. The obtained value will be used to dynamically update the initial threshold value. This threshold value is then used to limit the injection of new packets in order to maintain the network traffic to an acceptable level below the saturation point. Whenever traffic decreases then the fixed initial threshold value is used instead of the dynamically obtained value, as it imposes lower restrictions on the injection of packets allowed into the network. This allows the mechanism to dynamically adapt to changing network load and traffic. Also the newly calculated threshold value is used only if it leads to a more restrictive injection limitation policy, and is higher than the minimum needed to guarantee the possibility of injecting packets and therefore the elimination of starvation. All the empirical values will be determined via simulation and presented section 6.3.1. This mechanism also requires storing the previous m samples of the busy virtual channels count in order to

have a better calculation of the dynamic threshold. Whenever a new packet is injected, the count of the number of busy virtual channels is stored in this table. The threshold value is then obtained by averaging the table contents when the network is determined to be near the saturation point. Figure 6.1 provides an illustration of how the DRIL mechanism is implemented [70].

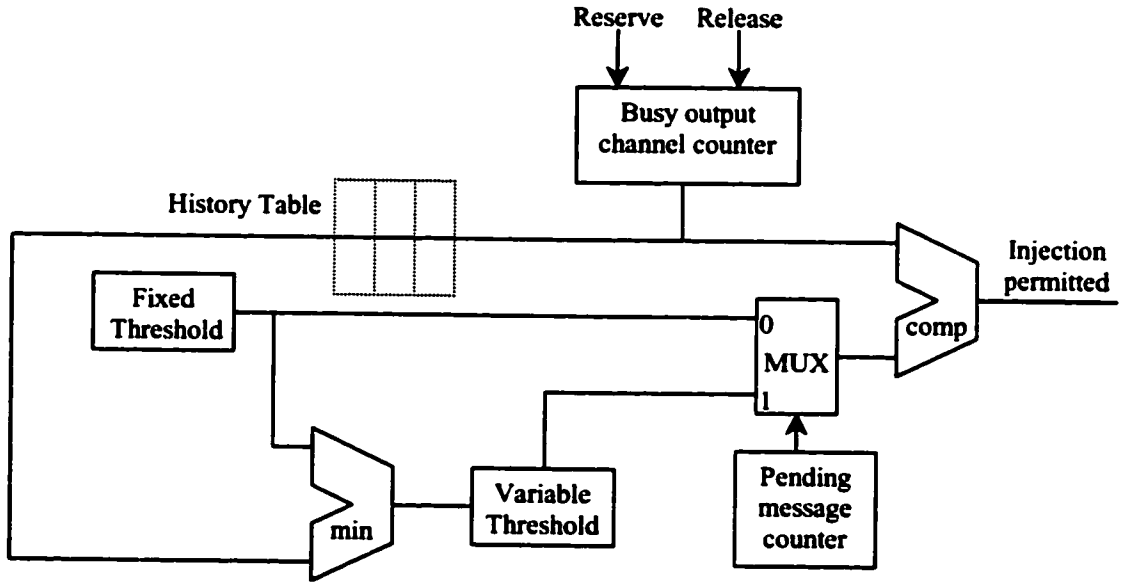


Figure 6.1. DRIL injection limitation mechanism

6.2 Congestion Level Injection Control (CLIC) Mechanism

The proposed injection limitation mechanism is a dynamic mechanism that estimates the network load by sensing the relative congestion of packets as calculated by each node in the network. The mechanism is referred to as *CLIC* (*Congestion Level Injection Control*). The CLIC mechanism was first published

by the authors in [71]. The operation of the CLIC mechanism is composed of distinguished components that are explained next:

6.2.1 Congestion Level Indicator (CLI)

Each node attempts to sense the congestion in the network by calculating the number of cycles consumed while routing a particular packet on each of its output channels. The objective of this feature is to associate a particular congestion value with each output channel of every node in the network. This value is referred to as the *Congestion Level Indicator* (*CLI*) value and it is more precisely calculated using a counter that is reset when the header of the packet is routed through the channel. The counter is then incremented at each clock cycle until the tail flit of that packet is finally passed through the channel. At this point, the value of the counter is committed to a register that is used to indicate the *CLI* value associated with the channel at any particular point in time. Output channels with high *CLI* values indicate that there is a relatively high level of congestion or traffic that exist ahead of that channel. If the *CLI* value is high enough then it may indicate that the network is approaching its saturation point. Since each channel is composed of several virtual channels that are used to route different packets by multiplexing them on the same physical channel. We elect to calculate the *CLI* value for only one of the virtual channels in order to reduce the hardware requirements. The *CLI* value for only one of the virtual channels is representative of the overall congestion level near that output channel because we are using true fully adaptive routing, where all virtual channels of a particular channel are used in a similar fashion without any restrictions or variations. Also the fact that all those similar virtual channels are multiplexed over the same channel, which factors the congestion level for all virtual channels within a single *CLI* value for a particular virtual channel. This mechanism requires as many counters as there are channels. Therefore four counters and registers are required in our particular network configuration. In this mechanism virtual channel number 0 that is associated with each physical channel is used to represent that channel. Also all the *CLI* values are initially set to the packet length in flits, as it is the

minimum possible value for a *CLI* in the absence of any contention or traffic from other packets in the network.

6.2.2 Injection Control

The injection control or limitation mechanism operates at the packet routing level. When a new packet is selected for injection from the head of the injection queue, the routing unit that implements the adopted routing algorithm processes the packet. The routing function generates a list of profitable output channels that the packet can be routed through. The selection function then selects one of those output channels that the packet should be injected on. The injection control logic is then invoked with this selected output channel in focus. If the *CLI* value for the selected channel is lower than a predetermined threshold T (initially set to T_{fixed}), then packet injection is allowed; otherwise if it has a higher value than the threshold then injection of the new packet is prevented during the current cycle. This logic is illustrated in Figure 6.2. The remaining logic shown in the flow chart is there to insure that the mechanism is dynamic in nature.

The mechanism attempts to more accurately calculate the *CLI* value at which the network is deemed to be near its saturation point by keeping track of the *CLI* value when packets are injected into the network. This is accomplished using the following formula:

$$C_i = \frac{(k-1)CLI_i + C_{i-1}}{k}$$

The value of C represents an indicator or average of all previous *CLI* values. It is an efficient method of acquiring an average of previous and recent *CLI* values without having to allocate many registers to hold several previous *CLI* values as implemented in the DRIL mechanism. The k variable used in the formula is referred to as the weight factor. The value of k used in the formula is 9, which

provides more weight to the recent value of the *CLI* on the overall *C* expression. This value is constant and does not change (for example if the value of k is to 2 then the expression becomes a simple averaging expression).

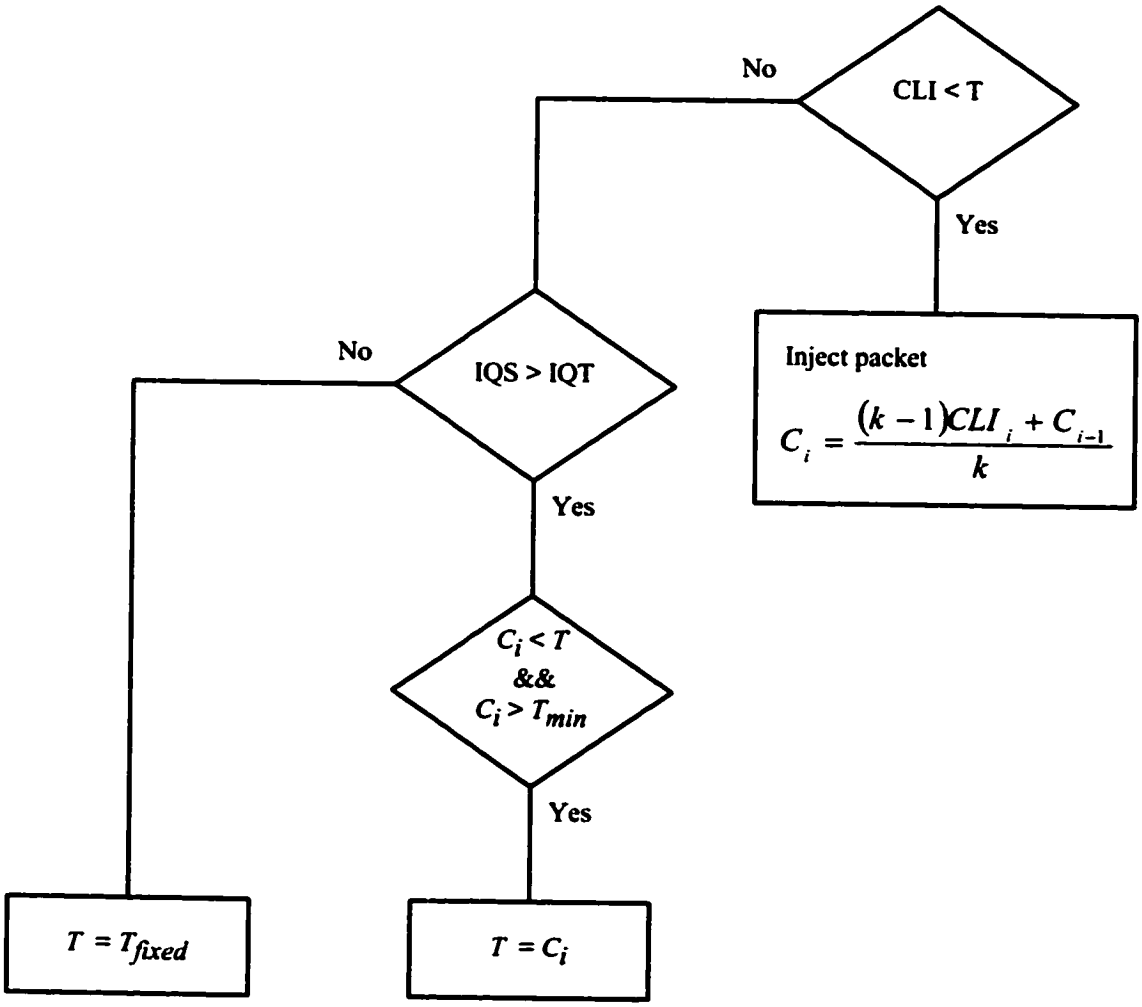


Figure 6.2. CLIC injection limitation mechanism.

When the network is suspected to be near its saturation point, as indicated by a predetermined number of packets that accumulate in the injection queue then the threshold is modified to be the latest calculated value of *C*. This is indicated by the *IQS* (*Injection Queue Size*) > *IQT* (*Injection Queue Threshold*) decision block. This branch is not executed unless the *C* value leads to a more restrictive injection policy, and it is above a predetermined minimum congestion threshold. This minimum threshold exists because certain levels of congestion counts are normal due to packet length and the existence of other traffic in the network. This value will be determined in section 6.3.2. Also this value is constant and not dynamic in nature. The last branch of the logic in the chart is there to insure that the mechanism is dynamic even if the network load has decreased. If that happens then the network is not close to its saturation point and the threshold value is reset to the initial predetermined threshold value, which is referred to as T_{fixed} and it allows for a less restrictive injection control policy.

6.2.3 **Injection selection function**

The Congestion Level Injection Control (CLIC) mechanism as detailed in the previous section prevents the injection of a packet on a requested output channel if the Congestion Level Indicator (CLI) value associated with that channel is higher than a predetermined threshold. This mechanism should be complemented by a packet selection function that selects the channel with the lowest *CLI* value amongst all the profitable channels passed to it by the routing function. This is to safeguard against the unnecessary restriction on packet injection in the presence of profitable channels that have lower and acceptable *CLI* values. This feature does not conflict with the *straight-first* selection function that is used throughout this effort. The straight-first selection function influences the channel selection decision that a packet in transit should be directed through, and not a newly injected packet. Injected packets have no orientation with regard to direction, as they have not been injected onto any channel yet. The channel selection policy for injected packets aside from the CLIC mechanism selects the first free output channel from the selection list.

6.3 Empirical Threshold Determination

This section is concerned with finding the various proper threshold values for both of the *DRIL* and the *CLIC* mechanisms so that their performance can be compared against each other. The threshold values are obtained empirically using simulation for all the traffic patterns used in our performance evaluation effort.

6.3.1 DRIL Thresholds

The *DRIL* mechanism depends mainly on the number of occupied virtual channels per node. The proper threshold value for the virtual channels count should be the number of virtual channels occupied just before the network enters into saturation. To determine this value using simulation, the average number of occupied virtual channels per node is plotted against the normalized load rate. Also to determine the number of packets that are pending in the injection queue when the network is deemed to be saturated, simulation is used to obtain the average injection queue size as a function of the normalized load rate. The injection queue size threshold will also be used for the *CLIC* mechanism. These two threshold values are plotted on the same charts. Figure 6.3 below shows the simulation results for both of these threshold variables plotted as a function of the normalized load rate for the uniform traffic pattern.

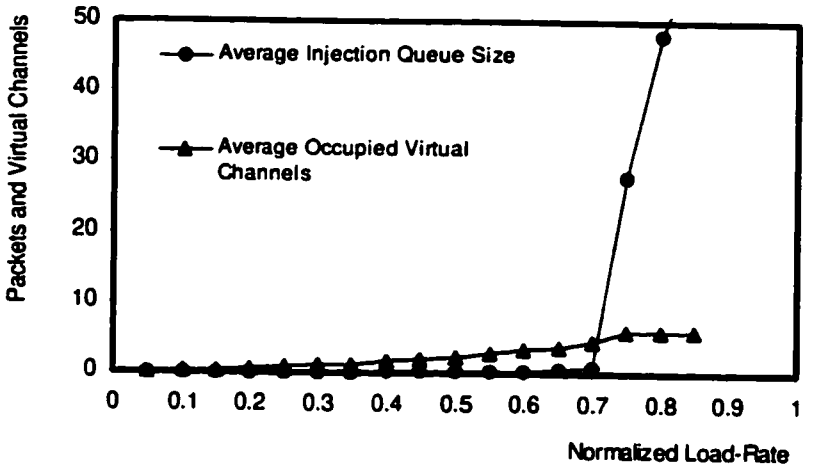


Figure 6.3. 2-dimension 16x16 mesh (Uniform traffic)

From the above chart we can determine that the network enters its saturation point at approximately 0.7 of the normalized load rate as was observed in our previous performance evaluation described in chapter 5. The number of occupied virtual channels threshold value for this traffic pattern is approximated to be 6, while the injection queue size threshold value is approximately 10. It is important to note here that the threshold values obtained from the chart are only approximate values. The proper threshold values should be obtained by actual simulation run trials. The chart provides minimum asymptotic values for these thresholds. Table 6.1 provides the actual threshold values as determined by the simulation trials. Similarly, Figures 6.4, 6.5, and 6.6 below display the simulation threshold values for the Bit-reversal, Dimension-reversal, and Hot-spot traffic patterns respectively. The proper threshold values for these traffic patterns as determined by simulation trials are also shown in Table 6.1 below.

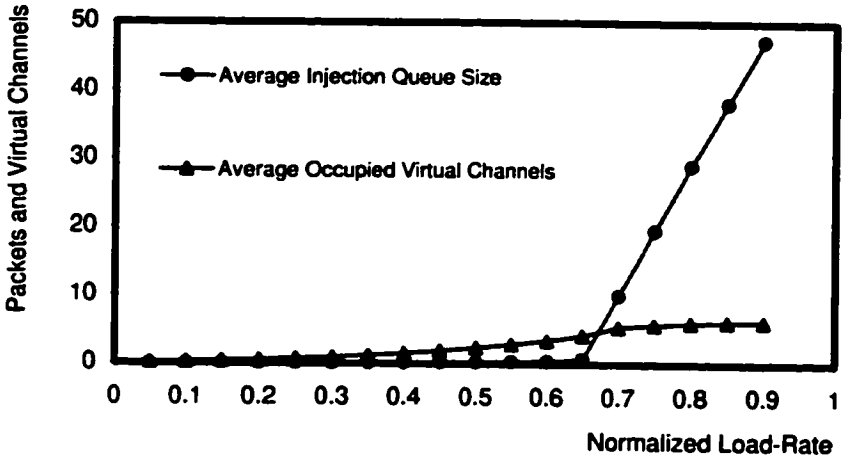


Figure 6.4. 2-dimension 16x16 mesh (Bit reversal traffic)

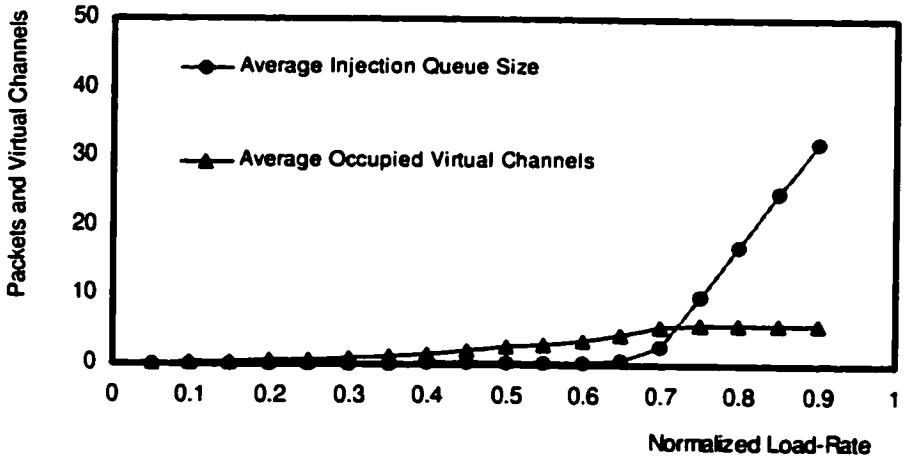


Figure 6.5. 2-dimension 16x16 mesh (Dimension reversal traffic)

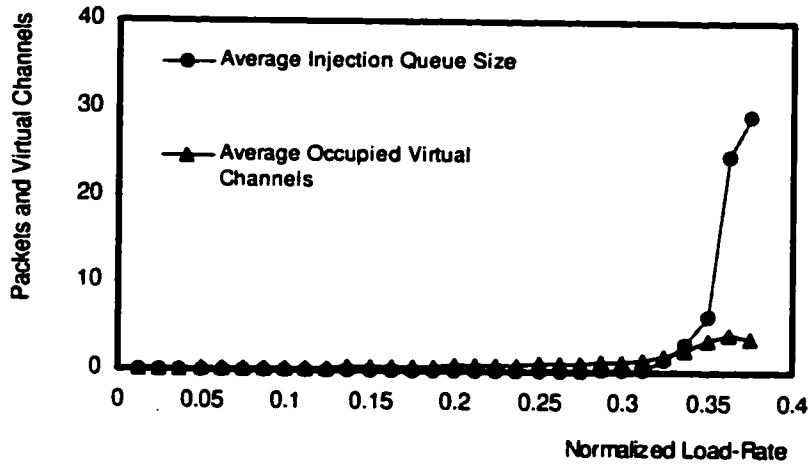


Figure 6.6. 2-dimension 16x16 mesh (Hot spot traffic)

6.3.2 CLIC Thresholds

The *CLIC* mechanism needs to establish a congestion level threshold that occurs in the network just before it enters into the saturation phase. This value is determined empirically by calculating the average value of those channels with the maximum congestion values per node in the network throughout a particular run. This threshold therefore represents an average of an average of the highest CLI values that occur in the network. Accordingly this threshold is referred to as the *Maximum CLI* threshold and it is plotted as a function of the normalized load rate.

Certain levels of registered congestion is normal because of the packet length and multiplexing delay due to the presence of other packets in the network. Therefore we also need to empirically determine this normal level of congestion as described by the *CLIC* mechanism. This threshold is determined by calculating the average congestion values for all channels of all nodes in the network throughout a particular run. This threshold value therefore represents an average of an average of the average CLI values in the network. This normal threshold value is referred to as the *Average CLI*

threshold and it is also plotted as a function of the normalized load rate. Figures 6.7, 6.8, 6.9, and 6.10 below show the plots of these threshold values in the case of uniform, bit-reversal, dimension-reversal, and hot spot traffic patterns respectively.

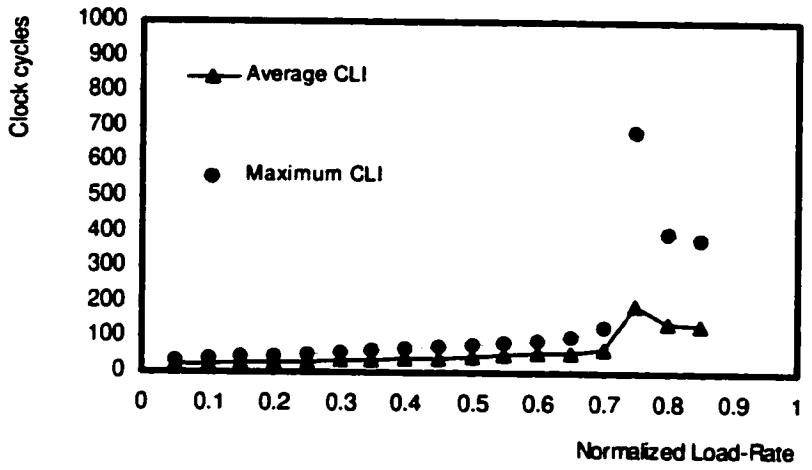


Figure 6.7. 2-dimension 16x16 mesh (Uniform traffic)

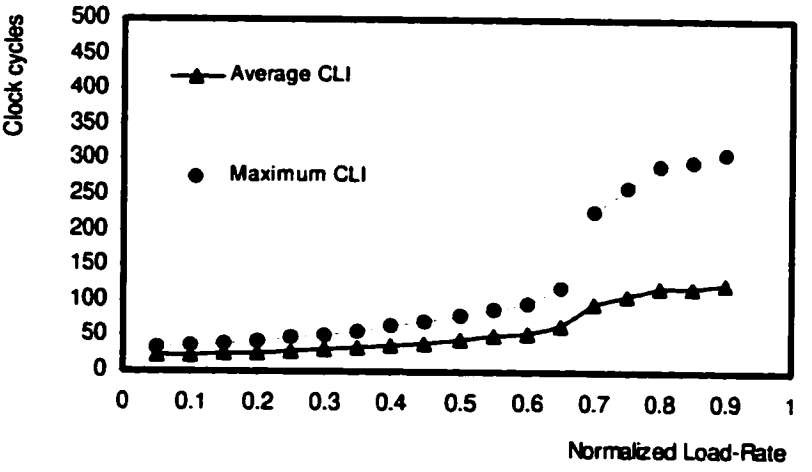


Figure 6.8. 2-dimension 16x16 mesh (Bit reversal traffic)

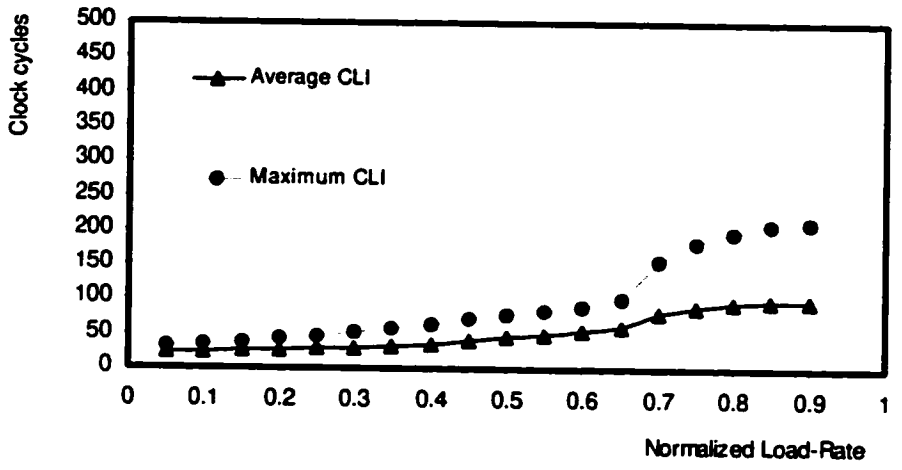


Figure 6.9. 2-dimension 16x16 mesh (Dimension reversal traffic)

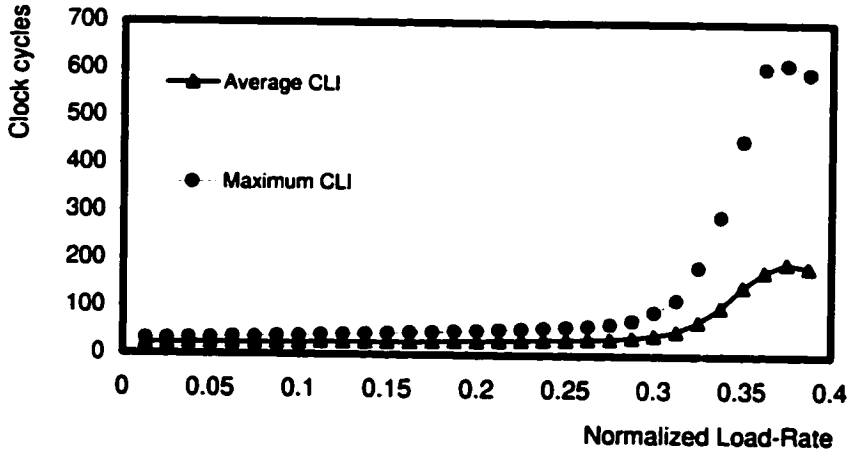


Figure 6.10. 2-dimension 16x16 mesh (Hot spot traffic)

It is important to note that these plots only provide asymptotic boundaries for the threshold values. The precise values of these thresholds still require some simulation trials in order to find their optimum levels. Table 6.1 below shows these threshold values that were determined via simulation, and were used in order to produce the simulation results of the next section.

TABLE 6.1 Injection Limitation Mechanisms Threshold Values

Threshold	Uniform	Bit Reversal	Dimension Reversal	Hot Spot
Busy Flit Buffers	8	9	11	4
Injection Queue Size	10	10	10	10
Congestion Threshold (T)	170	180	170	130
Minimum Congestion Threshold	65	60	120	32

6.4 Simulation Results

This section will present all the simulation results obtained using the *WormSim* simulator. The results compare the performance of the *CLIC* and the *DRIL* injection limitation mechanisms when used in conjunction with the *ZOMA* algorithm. The performance of the plain ZOMA algorithm as evaluated in chapter 5 is also displayed in order to measure the performance gains of the injection limitation techniques. The simulation results are obtained for the same multicomputer model and configuration as provided in sections 4.1, and 4.2 and the same traffic patterns detailed in section 4.2.2.

6.4.1 Uniform Traffic

The performance of the ZOMA algorithm along with the CLIC and DRIL injection limitation mechanisms are shown in Figures 6.11 and 6.12 below.

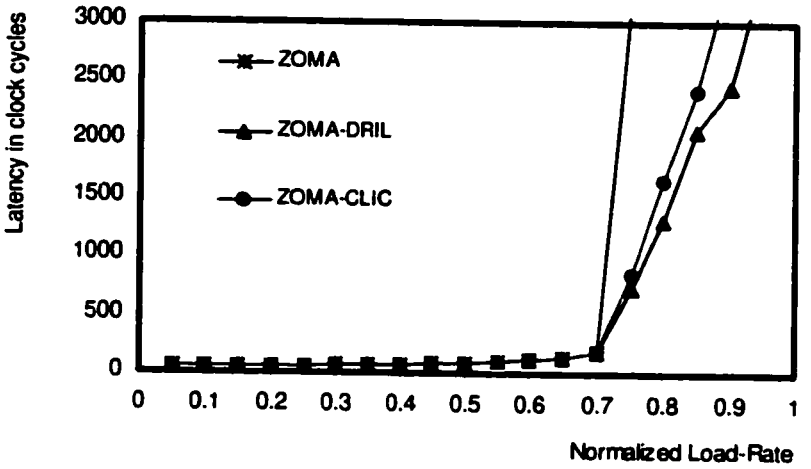


Figure 6.11. 2-dimension 16x16 mesh (Uniform traffic)

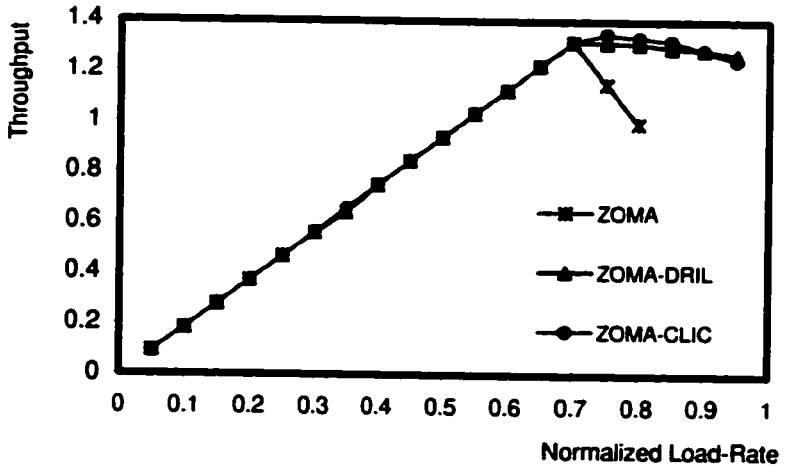


Figure 6.12. 2-dimension 16x16 mesh (Uniform traffic)

The performance charts show that major performance gains are achieved when applying the CLIC or DRIL injection limitation mechanisms over the plain ZOMA algorithm. The ZOMA algorithm, as shown in chapter 5, saturates around 0.7 of wire capacity. CLIC saturates around 0.85, while DRIL saturates around 0.9 of wire capacity. This corresponds to a 21% and 29% of performance improvement over the plain ZOMA respectively. In addition to the performance improvement demonstrated, the injection limitation mechanisms are effective in overcoming the steep performance degradation behavior displayed by the ZOMA algorithm in Figure 6.12. It is also worth mentioning that the latency figures shown here include the delay caused by source queueing of packets before their release into the network. Injection limitation techniques attempt to delay the injection of packets into the network in response to a perceived high level of traffic already in existence throughout the interconnection network. This results in increasing the source queueing component of the latency figures but improving the overall latency performance behavior. Some performance evaluation of injection limitation techniques therefore do not include the source queueing component in the latency figures, in which case the latency would remain somewhat constant similar to that demonstrated in the throughput figures [4]. The latency results shown here are therefore consistent with the latency results shown in chapter 5, because the same calculation method is adopted.

Reviewing Figure 6.11, we also notice that the DRIL mechanism demonstrates better performance results than the CLIC mechanism. The figure suggests that DRIL has a 6% performance improvement over CLIC. But carefully examining Figure 6.12 shows that CLIC demonstrates higher throughput characteristics than DRIL, which makes the direct comparison of both of these mechanisms nontrivial. In Figure 6.12 CLIC exhibits roughly about 2% higher throughput than DRIL. Also the performance gain in throughput and latency are not directly comparable. The injection limitation mechanism attempts to throttle or control the amount of traffic in the network in order to allow for the smooth progress of packets throughout the network. In other words these mechanisms attempt to balance the amount of traffic or throughput delivered by the network as opposed to the delay or latency incurred by the packets that are being routed through the network. A good injection limitation mechanism should attempt to maximize the achievable throughput in relation to the obtained latencies. Therefore in order

to evaluate the performance of these injection limitation mechanisms, both of these metrics have to be considered simultaneously. Experimental observation shows that even slight improvements in throughput are more difficult to achieve than the corresponding latency metric. Empirical results suggest that a 1% improvement in throughput corresponds roughly to about a 10% improvement in the latency figure. If we calculate the various latency figure improvements between the two injection limitation mechanisms, we arrive at an average of 18% improvement in latency for DRIL over CLIC. Weighing that against a 2% throughput enhancement for CLIC over DRIL, we can arrive to the conclusion that the overall performance characteristics of both of these injection limitation mechanisms are somewhat similar. But we should also tip the performance characteristics slightly in favor for DRIL especially at high load rates. We can explain this observation as follows. The random traffic pattern demonstrates sharp saturation curves as illustrated in Figures 6.5, 6.7, 6.11, and 6.12. This is due to the cyclic routing behavior as explained in section 5.5.2.1. Presented with this kind of traffic pattern, a strict injection limitation policy such as the one adopted by DRIL maybe more effective in avoiding saturation especially at high load rates. This is because when DRIL detects a high number of busy virtual channel buffers, it prevents the entire node from injecting any packets. While in CLIC each channel individually detects the high level of congestion ahead of it and prevents the injection of packets on that channel only. Other channels that are connected to the same node may not yet detect the same level of congestion and therefore will continue to inject packets into the network. This behavior by its design will have a slower reaction than that in DRIL in response to a high level of traffic in the network, but it allows for the network to accept more traffic. This explains the behavior that CLIC demonstrates higher throughput but somewhat higher latency figures. This observation will also be used to explain the behavior of the other traffic patterns used in the next section.

6.4.2 Non-Uniform Traffic

This section will present the simulation results for the different non-uniform traffic patterns that were explained in section 4.2.2. Namely the bit-reversal, dimension-reversal, and the hot spot traffic patterns.

Bit-reversal traffic. Using the *bit-reversal* traffic pattern, the simulation results obtained are shown in Figures 6.13, and 6.14 below.

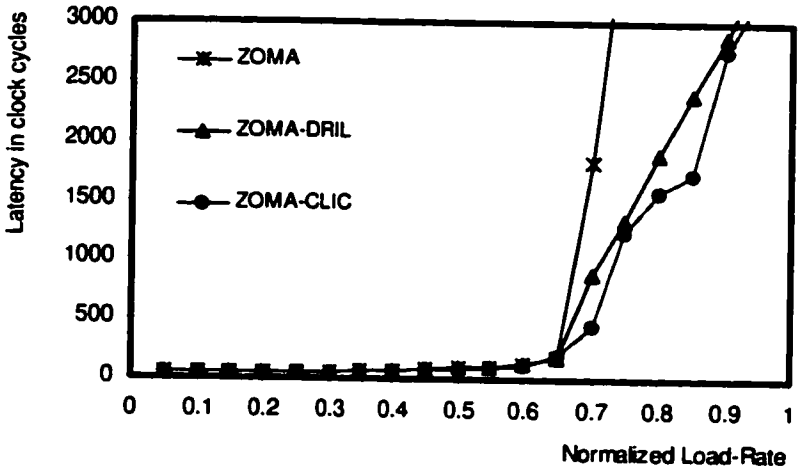


Figure 6.13. 2-dimension 16x16 mesh (Bit reversal traffic)

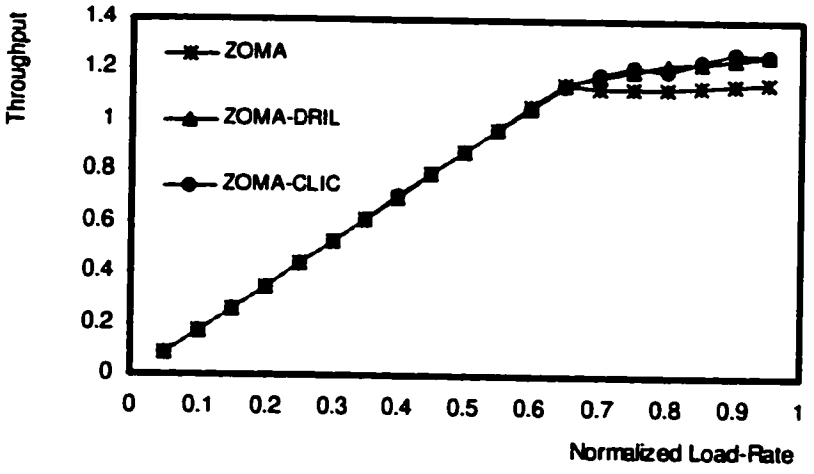


Figure 6.14. 2-dimension 16x16 mesh (Bit reversal traffic)

The following observations can be made by analyzing the performance charts. First, the performance improvement of the injection limitation mechanisms is substantial. Second, the CLIC mechanism outperforms the DRIL mechanism in this non-uniform traffic pattern in contrast to what was observed earlier in the uniform traffic pattern. CLIC saturates approximately around 0.9, while DRIL saturates around 0.85, and plain ZOMA saturates around 0.65 of the normalized load rate. This corresponds to a 6% and 38% of performance gain for CLIC over DRIL and the plain ZOMA respectively. It was suspected that CLIC would outperform the DRIL mechanism for the non-uniform traffic pattern cases, but we need to insure that this is the case for the remaining non-uniform traffic patterns. This matter will be discussed in more detail in the next traffic pattern case.

We can also notice from Figure 6.14 that the throughput performance is improved over the plain ZOMA, just as in the uniform traffic pattern, although the bit-reversal traffic pattern does not actually demonstrate sharp performance degradation beyond saturation in the plain ZOMA case. This is due to the previously mentioned behavior that this traffic pattern does not lend itself to forming cyclic dependencies while routing. This means that these injection limitation mechanisms improve the performance of the plain ZOMA algorithm even for those traffic patterns that do not cause serious performance degradation beyond the saturation point. More importantly, as can be observed by the chart, the CLIC mechanism achieves slightly higher throughput than DRIL, even that it already demonstrates better latency results. This makes the performance characteristics of CLIC even more favorable than those displayed by DRIL. These observations will be elaborated on in more detail in what follows.

Dimension-reversal traffic. In the *dimension-reversal* traffic pattern case, the simulation results are shown in Figures 6.15, and 6.16 below.

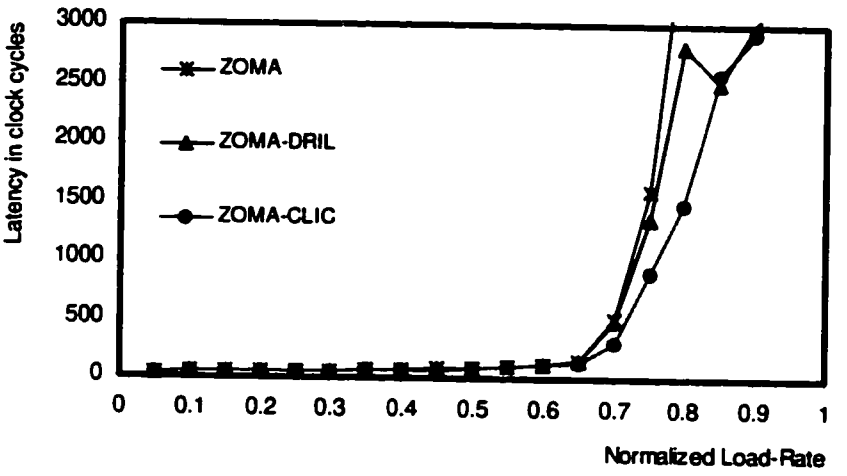


Figure 6.15. 2-dimension 16x16 mesh (Dimension reversal traffic)

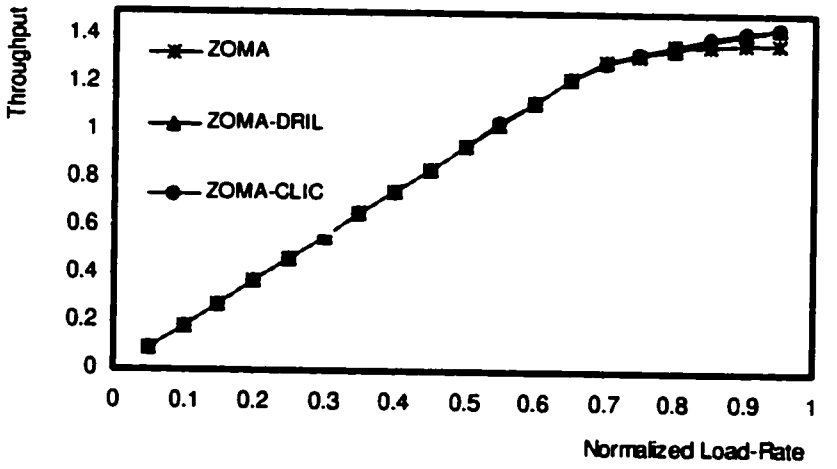


Figure 6.16. 2-dimension 16x16 mesh (Dimension reversal traffic)

The charts above show that the CLIC mechanism has a clear performance advantage over the DRIL mechanism, and that both of them outperform the plain ZOMA algorithm. Plain ZOMA saturates around 0.7, DRIL around 0.8, while CLIC saturates at around 0.9 of wire capacity. This corresponds to about a 29% and a 13% performance gain for CLIC over plain ZOMA and DRIL respectively. Figure 6.16 shows that the throughput is not only generally improved at high load rates using these

mechanisms, but more specifically the throughput achieved by CLIC is even slightly higher than that demonstrated by the DRIL mechanism. This edges the performance gain of CLIC even higher as discussed in the previous sections. These findings and the previous results of the bit-reversal traffic pattern confirm the trend that CLIC outperforms DRIL for the non-uniform traffic patterns for the same reasons that the latter outperformed CLIC for the uniform traffic pattern. More precisely it is the fact that CLIC senses the abnormally high congestion per each individual channel of each node, which allows each node to effectively shut down or continue injection on different channels depending on the state of the network near those channels. Non-uniform traffic patterns, which more closely resemble the behavior of real parallel applications, tend to concentrate traffic non-uniformly throughout the network and in different areas and pockets of that network. This causes the channels to be loaded differently and variably. This suggests that manipulating the injection of packets on a more granular level than the node unit, which is the channel unit in this case can lead to achieving much higher performance characteristics as the simulation results have confirmed.

There is yet another unfavorable characteristic that can be observed about the DRIL mechanism. Figure 6.15 obviously shows that the performance of DRIL is inconsistent as evident by the anomaly in the latency graph around the 0.85-loading rate. This behavior could not be avoided using different thresholds without seriously sacrificing the performance gains of the DRIL mechanism. Also this trend will be confirmed by the hot spot traffic pattern as well, so it is not transient or just due to random number fluctuation errors. The explanation for this problem is offered here as this same behavior was experienced in the early stages of developing and refining the CLIC mechanism, and is one of the main reasons that led to modifying the CLIC mechanism to its current form. Basically when we have a rough injection control mechanism, such as DRIL, the whole node is prevented from injecting packets into the network when a certain threshold is reached. If the ongoing recalculations of the variable against which this threshold is compared against are not dynamic enough, it may lead to long periods of dormant node behavior. When this occurs throughout the network it can collectively disturb the obtained results into being inconsistent or can even more drastically lead to the starvation of certain nodes in some cases.

The starvation possibility was actually mentioned in [70], but nor its effects on the stability of the performance results and neither a solution for the problem were offered in the paper. In order to solve this problem, extra hardware is needed in the form of timeout counters and registers. The function of these counters is to reactivate a particular node if it has been dormant longer than a predetermined timeout value regardless of the injection limitation threshold value. Again, this hardware was implemented and modeled in the early stages of developing the CLIC mechanism, and was found to be the only mean of overcoming this problem. The authors of the DRIL mechanism did not include this extra hardware in their logic, as was detailed in section 6.1.1. This problem is avoided in CLIC by having the injection control mechanism operate at the channel level. This provides ample flexibility to the node as it is allowed to control the injection of packets along different channels. Only when the node senses high congestion on all of its outgoing channels does it become completely dormant. But when that happens, the node can also be activated much quicker whenever it senses a moderate level of congestion on any of its channels. This provides CLIC with the necessary dynamics to bypass this problem.

Hot-spot traffic. The *hot-spot* case had been previously shown to be the most difficult traffic pattern for the network to manage. The results using the injection limitation techniques are shown in Figures 6.17 and 6.18 below.

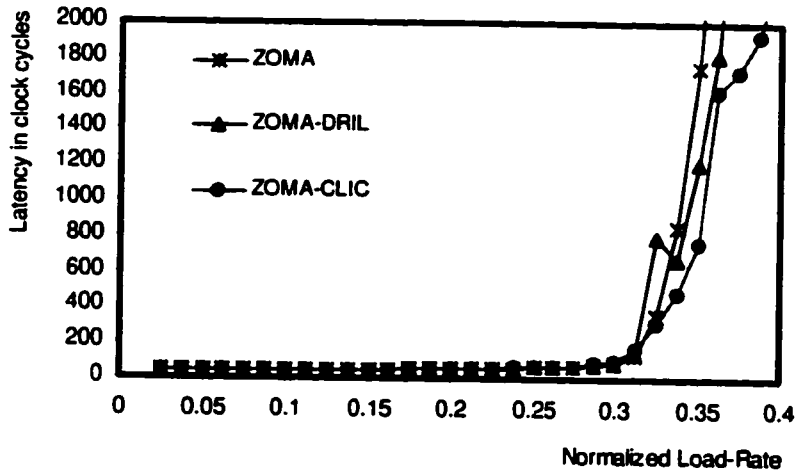


Figure 6.17. 2-dimension 16x16 mesh (Hot spot traffic)

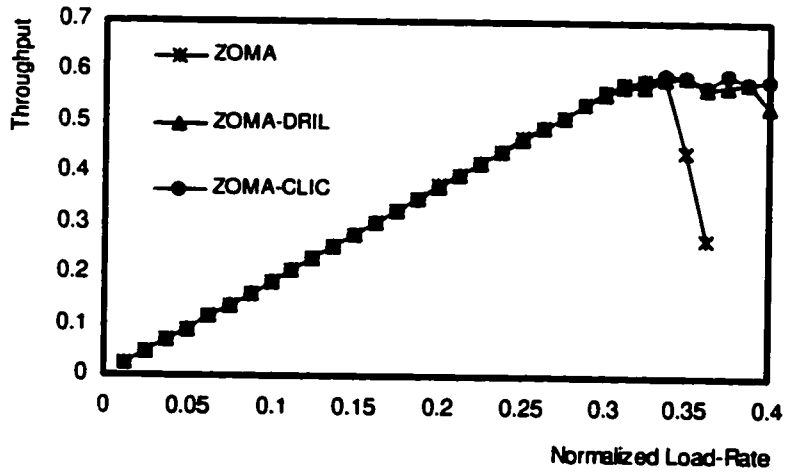


Figure 6.18. 2-dimension 16x16 mesh (Hot spot traffic)

The charts show that CLIC outperforms DRIL on both accounts of latency and throughput. The performance patterns demonstrated by both injection limitation mechanisms, as a result of the high congestion caused by this traffic pattern, makes determining their saturation point somewhat more challenging. Taking the throughput figure into consideration as well as that of latency, it is

approximated that CLIC saturates at around 0.375, while DRIL saturates at 0.35, and plain ZOMA saturates at 0.3375 of the normalized load rate. This provides CLIC with a 7% and 11% performance improvement over DRIL and plain ZOMA respectively. The performance advantage is more apparent when observing the throughputs achieved by the two mechanisms. CLIC demonstrates higher and more stable throughput behavior than DRIL. This is in line with previous observations that CLIC outperforms DRIL for the non-uniform traffic patterns and for the same reasons mentioned earlier as well. Namely the finer level of packet injection control down to the channel rather than the node level, which seems to be more suitable for the nature of the non-uniform traffic patterns and more significantly, traffic patterns that are exhibited by real parallel applications.

Also the performance figures confirm the previously made notion that the performance obtained by the DRIL mechanism is inconsistent as apparent by the behavior observed at 0.325 of normalized load rate. At and around that point, the performance of DRIL is surprisingly even worse than that of the plain ZOMA algorithm. Experimenting with various other DRIL threshold values did not eliminate this behavior, but rather worsened it. In fact what is shown is the best possible performance behavior that could be achieved using DRIL. This behavior can be explained by the paranoiac reaction of preventing the entire node from injecting any packets, while only certain channels of the node are being overloaded. To support this argument, it can be further observed that the throughput provided by DRIL is lower than that provided by the plain ZOMA algorithm at that mentioned traffic rate. This confirms that DRIL allowed less traffic to be injected into the network as did the plain ZOMA algorithm, which led in this case to higher latencies as a result of the unnecessarily delays in packet injections. In addition to this weak performance, DRIL is not successful either in totally eliminating the degraded performance behavior that occurs after reaching the saturation point, as evident by the dip in throughput at high load rates. This again adds more performance praise to the CLIC mechanism.

Another essential function of an injection limitation mechanism, especially as it is used along true fully adaptive routers, is its ability to reduce the frequency of deadlock occurrences throughout the

network. To assess the performance of CLIC in this regard, the frequency of deadlocks detected by the two injection limitation mechanisms is recorded and compared against that obtained earlier in chapter 5 for the plain ZOMA algorithm. This is obtained for that traffic distribution that causes more deadlocks to occur, which is namely the hot spot traffic pattern. The following chart shows this comparison. More specifically what is shown is the number of deadlocks detected normalized to the total number of packets delivered by the network for the various load rate factors.

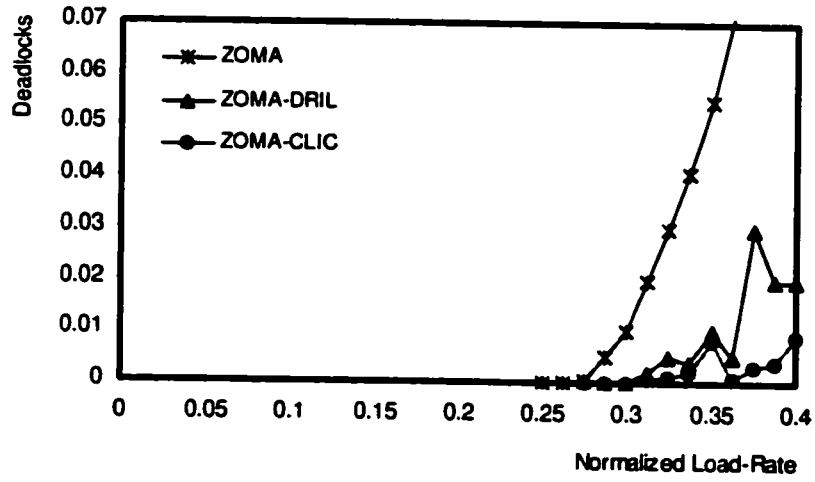


Figure 6.19. Frequency of deadlocks in hot spot traffic

From the chart it is clear that the CLIC injection limitation mechanism also provides the best performance with regard to the frequency of deadlocks detected. As can be seen both injection limitation mechanisms are able to reduce the frequency of deadlocks detected, with CLIC providing the lowest number of deadlocks detected in the network. This is in spite of the fact that CLIC provides the highest throughput for all traffic patterns evaluated when compared with either the DRIL mechanism or the plain ZOMA algorithm. This combination is significant since the CLIC mechanism did not reduce the frequency of deadlocks by the reducing the overall traffic circulating around the network. Rather it

allowed more traffic to go through the network, and was able to deliver this higher payload with less number of deadlocks. Traffic and therefore cyclic dependencies gradually increase throughout the network, which causes a comparable increase in the number of detected deadlocks. When traffic exceeds the predetermined threshold levels, the injection limitation mechanism is activated. This occurs around 0.35 of the normalized wire capacity. The injection limitation process reduces the level of traffic, cyclic dependencies, and consequently the number of deadlocks, which is evident by the dip at the 0.3625 normalized rate. From that point on, CLIC gradually increases the traffic with an accordingly manageable number of deadlocks. DRIL on the other hand does not demonstrate gradual and smooth increase in traffic, as demonstrated by the sharper rise in number of deadlocks, which affect the performance of the entire network and eventually cause degradation in performance that can be observed at the 0.3875 normalized load rate, and this can be verified using Figure 6.18 and observing the dip in throughput at that loading factor. This again highlights the smart logic of the CLIC injection limitation mechanism that is due to controlling the injection at the individual channel level. This can be attributed to the following analysis. By sensing individual channels and limiting the injection on already congested channels, there are fewer tendencies for cyclic dependencies to form amongst occupied channels and channel resources. Cyclic dependencies in turn are precursors to deadlock formations, so the problem of deadlocks is being tackled at its root cause, which is an effective remedy as illustrated by the deadlock frequency chart.

The results of this chapter illustrated the superior performance of the CLIC mechanism over DRIL for the non-uniform traffic patterns, while the latter had a slight performance advantage when the random traffic pattern was applied. CLIC demonstrated that it had well achieved the two most important objectives of any injection limitation mechanism stated at the beginning of the chapter. First, CLIC was able to maintain the performance stability of the network at high load rates, and it was able to eliminate any severe performance degradation behavior near or beyond the saturation point. Second, CLIC was well capable of reducing the frequency of deadlocks detected in the network, by preventing

the network from reaching the state where cyclic dependencies start to frequently form due to high congestion. These two significant features of an injection limitation mechanism make low-cost and efficient deadlock-recovery techniques, such as ZOMA, more viable and sensible alternatives. CLIC successfully achieves these two objectives as was demonstrated in the simulation performance results.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

This thesis was motivated in part by the growing reliance on the class of Massively Parallel Processors in order to solve cumbersome and time consuming problems that would otherwise seem intractable in reasonable time. This motivation along with the precise statement of the problem and achievements of this effort were clearly stated in the beginning of this thesis.

A thorough review of the popular wormhole switching technique that is used for multicomputer interconnection networks was provided. Wormhole switching was widely adopted in multiprocessors due to its favorable performance and low cost properties. Interconnection networks, being the most significant component affecting the performance of message-passing multicomputers, were discussed and classified into direct and indirect interconnection networks. This thesis primarily considered the class of direct interconnection networks for their merits, which were highlighted. Also the major characteristics of direct networks, namely their topology, switching techniques, flow control, and routing where then considered in depth. Various important definitions and concepts that are closely related to wormhole switching and routing were elaborated on. These definitions include the deadlock issue, livelock, starvation, and the virtual channel concept.

An in-depth overview of the routing algorithms was provided, as routing is a very important characteristic of the underlying switching mechanism. The routing algorithm utilized by the switching mechanism determines how packets are routed over the interconnection network and strongly influences the performance of that network. A classification and characterization of the routing algorithms according to the amount of adaptivity they provide and the approach they use to overcome the issue of deadlock was provided. Various frameworks that can be used to create deadlock-avoidance routing algorithms were covered. Deadlock-recovery routing algorithms were also categorized into progressive and regressive algorithms. A survey of the most representative routing algorithms used in the literature

and their detailed method of operation was provided. A cost model that can be used to directly compare the hardware requirements of these routing algorithms was discussed and used to evaluate the router architectures of some of these algorithms.

The assumptions made about the network model, its topology, configuration parameters, router architecture, message model, traffic patterns and generation method were provided. Details about the performance metrics used to evaluate the proposed routing algorithm and mechanism against those from the literature were given. The details and the structure of the *WormSim* simulator that was developed as part of this thesis were provided. The performance evaluation of all the selected routing algorithms and proposed mechanisms in this thesis was achieved via simulation, utilizing the WormSim simulator.

This thesis proposed a true fully adaptive routing algorithm with a new deadlock-recovery mechanism named ZOMA. The ZOMA deadlock-recovery algorithm is efficient and takes advantage of the concept of wormhole switching in terms of low hardware resource requirements. The performance of the ZOMA algorithm was on par with other more expensive deadlock-recovery mechanisms such as Disha, while requiring lower hardware resources that are not on the critical path of switching normal packets as demonstrated by the cost model evaluation of both algorithms. ZOMA belongs to a new category of deadlock recovery techniques that we referred to as preemptive as opposed to the existing progressive and regressive categories. Although the Duato algorithm demonstrated slightly better performance outcome in comparison to ZOMA for one of the traffic patterns, these results are inconsistent for all traffic patterns, network topologies, and configurations. For example, Duato displays lower performance in torus networks. A major disadvantage of the Duato algorithm is that it cannot be applied in network configurations that do not meet its virtual channel requirements. ZOMA algorithm is more consistent and generally applicable to a wider class of network topologies and configurations. Results also showed that ZOMA caused fewer deadlocks to be detected as was demonstrated in the hot spot traffic case. This was attributed to the fact that ZOMA acted as a faster draining mechanism for deadlocked packets. By quickly removing these deadlocked packets from the network and releasing their resources simply prevented more deadlocks from forming behind them. ZOMA could be even more efficient if applied within modular router designs such as the enhanced-

hierarchical router that will most likely be used in future router architectures. Disha requires an additional port in each subcrossbar, which will make it more expensive as demonstrated by the cost model. Also when using a router model that is input and output buffered, there will be no extra hardware needed by ZOMA. While Disha suffers from this problem, as output buffers will have to be cleared, and stored in additional hardware during the progressive deadlock recovery phase. This extra overhead was not modeled in this thesis as input buffering was assumed throughout the simulation effort.

The thesis also proposed a new injection limitation mechanism called CLIC. The CLIC mechanism was used to complement the ZOMA deadlock-recovery algorithm and boost its performance. CLIC protects against excessive traffic in the network by locally calculating and controlling the level of congestion experienced along each individual channel by those packets that are traveling through it at a particular moment. The CLIC mechanism successfully achieved the objectives it was designed to meet. Namely, it eliminated the severe performance degradation behavior observed when the particular traffic pattern was susceptible to cyclic dependencies. Results demonstrated that CLIC allowed these networks to maintain their performance stability beyond their earlier observed saturation points. Also CLIC was able to effectively reduce the frequency of deadlocks that tend to occur near or beyond the saturation point. Not only was CLIC capable of achieving these functions, but it achieved them with more favorable performance results than other injection limitations mechanisms, such as DRIL. CLIC also clearly demonstrated superb performance improvements over DRIL in almost all traffic pattern cases and performance metrics. The excellent performance of CLIC was attained because of its design. More specifically exercising injection limitation at the channel level rather than the node level allowed CLIC to achieve superior performance in all traffic patterns that load the network non-uniformly resembling the behavior of real parallel applications. The CLIC mechanism therefore paves the way for true fully adaptive routing algorithms with efficient, low-cost deadlock-recovery mechanisms, such as ZOMA to be even more widely viable and applicable to wormhole networks.

For future work in this field of research there are various areas that could be investigated, which can

lead to further positive refinements and improvements to the proposed algorithms and mechanisms presented. For example it would be worthy to evaluate the ZOMA and the CLIC mechanisms for different network dimensions such as three and four-dimensional networks. True fully adaptive routing algorithms should perform better in comparison to other types of routing algorithms in higher dimensional networks as these networks provide the true and fully adaptive algorithm with more channel choices at each node to select from. This enables the TFAR algorithm, such as ZOMA to spread the traffic more evenly amongst the available channel resources within different dimensions, which can therefore lead to higher performance results.

Different network topologies, such as Torus networks, could also be investigated. This again would provide the ZOMA algorithm with more of a performance advantage, as traffic in tori tends to get equally distributed throughout the network, and not congested near the center as occurs in mesh networks. This is due to the wraparound channels that exist in tori and give the topology its fully symmetrical feature. Also algorithms, such as Duato, that demonstrated slight performance advantage over ZOMA in certain traffic patterns would give up that slight advantage because they would have to dedicate more virtual channel resources for their escape channels, while ZOMA would still utilize all the available virtual channel resources to adaptively service packets.

Evaluating various other network configuration parameters and investigating their effects on the performance of the proposed mechanisms would also be in order. These network configuration modifications could be the number of virtual channels used in the network, the depth of each virtual channel, the length of packets used, and so on. Also different traffic distributions could be studied and evaluated. The number of injection and delivery channels used at each node is also an important factor with an associated set of pros and cons that deserve to be further investigated. For example, higher number of delivery channels can alleviate the early saturation problem experienced in highly irregular traffic patterns such as the hot-spot case. But, while increasing the number of injection channels can lead to achieving higher throughputs, it may at the same time cause algorithms to reach early saturation points. Other more fundamental changes to the router architecture could also be investigated, such as the effect of using input-output buffered nodes. This was alluded to earlier, and is poised to provide

higher performance gains for ZOMA over Disha as explained previously.

Smarter misrouting techniques should also be well investigated as they hold a promise for well complimenting ZOMA. For example ZOMA can employ smart misrouting in the following fashion. When the header of a packet has been preempted and currently residing in the central buffer. We can elect to mark that header to be eligible for misrouting to another channel, especially if the remaining path to the destination node does not have any adaptivity left in it. In this case we would misroute the packet to another channel that is not on its shortest paths list. This would result in possible significant improvements gains for ZOMA, as the preempted packet can bypass a highly congested channel via a less utilized one, which would greatly reduce its overall latency caused by waiting on the occupied channel for longer periods of times.

Another area of research that might lead to performance improvements for the ZOMA algorithm is the parallel recovery of multiple deadlocked packets whenever they occur. This would require the duplication of the hardware required for performing the current deadlock recovery by a number of times according to the desired number of deadlocked packets to be recovered from simultaneously. Though this extra hardware may not be used efficiently, as it will be idle most of the time, and used in rare circumstances when the network is well beyond its saturation point.

There is another significant application for the Congestion Level Indicator (CLI) counters that are used as part of the CLIC mechanism. These counters could be utilized in order to create a new selection function. As defined previously, a selection function selects a single channel from the set of possible channels generated by the routing function in order to route the packet through it. The suggested selection function will select from the possible set of channels that channel with the lowest CLI value associated with it in order to route the packet. Research has shown that no single set of selection functions provided superior performance consistently for all conditions and traffic patterns. Moreover, tuning the selection function to the applied workload can significantly improve the performance. The suggested future work proposed here attempts to solve this problem by enabling the selection function to always select the output channel with the least congestion that may be presented to the packet as it travels through that channel. The implementation of this selection function may yield substantial

performance improvement results in addition to what has been demonstrated by the CLIC mechanism.

Finally, we could implement new features and enhancements to the WormSim object-oriented simulator. An important feature that could be added to WormSim is a graphical visualization interface, which can be relatively easily implemented with Java due to its well-developed graphical extensions. Visualization is an extremely valuable tool for any simulation effort, especially at the initial development phases of the work as it facilitates debugging and verification. This visualization front-end interface will provide a graphical representation of the network being simulated, and provides an intrinsic and dynamic view of the state of the network, congestion areas, channels, and resources in general. In essence, this graphical tool will be able to provide a detailed picture of any particular routing algorithm or mechanism and how this mechanism is behaving as the simulation progresses. This tool could be very effective in debugging, testing, and applying improvements to any particular routing algorithm or mechanism under investigation.

Nomenclature

ATM	Asynchronous Transfer Mode
BMIN	Bidirectional Multistage Interconnection Network
CARP	Compiler Aided Routing Protocol
ccNUMA	Cache-coherent Nonuniform Memory Access
CLI	Congestion Level Indicator
CLIC	Congestion Level Injection Control
CLRP	Cache-like Routing Protocol
CNF	Chaos Normal Form
DOR	Dimension-Order Routing
DRIL	Dynamically Reduced message Injection Limitation
DSM	Distributed Shared Memory
FCFS	First Come First Serve
FIFO	First In First Out
FLITS	FLow control DigITS
IQS	Injection Queue Size
IQT	Injection Queue Threshold
KSR	Kendall Square Research
LAN	Local Area Network
LCG	Linear-Congruential Generator
LOC	Lines of Code
MAN	Metropolitan Area Network

MIN	Multistage Interconnection Network
MIT	Massachusetts Institute of Technology
MPP	Massively Parallel Processors
MRC	Mesh-Routing Chips
NOW	Network of Workstations
PAR	Planar-Adaptive Routing
PEC	Packed Exponential Connections
PHITS	PHysIcal Transfer unitS
SAN	System Area Network
SPC	Scalable Parallel Computers
SPIDER	Scalable Point-to-point Interface DrivER
TFAR	True Fully Adaptive Routing
VLSI	Very Large Scale Integration
WAN	Wide Area Network
ZOMA	Zaki Obaidat Mulhem Al-Bassam

BIBLIOGRAPHY

[1] Lionel M. Ni, Yadong Gui, Sherry Moore, "Performance Evaluation of Switch-Based Wormhole Networks," *IEEE Transactions on Parallel And Distributed Systems*, Vol. 8, No. 5, pp. 462-474, May 1997.

[2] Xiaola Lin, Philip K. McKinley, Lionel M. Ni, "Deadlock-Free Multicast Wormhole Routing in 2-D Mesh Multicomputers," *IEEE Transactions on Parallel And Distributed Systems*, Vol. 5, No. 8, pp. 793-804, August 1994.

[3] William J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, Vol. 39, No. 6, pp. 775-785, June 1990.

[4] Jose Duato, Sudhakar Yalamachili, Lionel Ni, "Interconnection Networks: an Engineering Approach," *IEEE Computer Society Press*, 1997.

[5] Craig B. Stunkel, "Commercially Viable MPP Networks," *In proceedings of the 1996 International Conference on Parallel Processing Workshop*, pp. 52-63, 1996.

[6] Anant Agarwal, "Limits on Interconnection Network Performance," *IEEE Transactions on Parallel And Distributed Systems*, Vol. 2, No. 4, pp. 398-412, October 1991.

[7] F. Silla, M. P. Malumbers, J. Duato, D. Dai, D. K. Panda, "Impact of Adaptivity on the Behavior of Networks of Workstations under Bursty Traffic," *In Proceedings of the 1998 International Conference on Parallel Processing*, pp. 88-95, 1998.

[8] F. Silla, J. Duato, A. Sivasubramaniam, C. R. Das, "Virtual Channel Multiplexing in Networks of Workstations with Irregular Topology," *In Proceedings of the Fifth International Conference on High Performance Computing*, pp. 147-154, 1998.

[9] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, Wen-King Su, "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, pp. 29-36, February 1995.

[10] Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, Charles P. Thacker, "Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links," *IEEE Journal on Selected Areas in Communications*, Vol. 9, No. 8, pp. 1318-1335, October 1991.

[11] Robert Felderman, Annette DeSchon, Danny Cohen, Gregory Finn, "ATOMIC: A High-Speed Local Communication Architecture," *Journal of High Speed Networks*, pp. 1-21, 1994.

[12] B. Horst, D. Avresky, R. Wilkinson, D. Jewett, W. Watson, L. Young, C. Cunningham, "Performance Modeling of ServerNet Topologies," *In Proceedings of the 1996 International Parallel Processing Symposium*, pp. 518-523, 1996.

[13] F. Silla, J. Duato, "Improving the Efficiency of Adaptive Routing in Networks with Irregular Topology," *In Proceedings of the 1997 International Conference on High Performance Computing*, pp. 330-335, December 1997.

[14] Lionel M. Ni, Philip K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *IEEE Computer*, Vol. 26, No. 2, pp. 62-76, February 1993.

[15] William J. Dally, Hiromichi Aoki, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels," *IEEE Transactions on Parallel And Distributed Systems*, Vol. 4, No. 4, pp. 466-475, April 1993.

[16] Albert Y. H. Zomaya, "Parallel & Distributed Computing Handbook," *McGraw-Hill*, 1996.

[17] Daniel H. Linder, Jim C. Harden, "An Adaptive Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes," *IEEE Transactions on Computers*, Vol. 40, No. 1, pp. 2-12, January 1991.

[18] Farooq Ashraf, "Routing in Multicomputer Networks: A Classification And Comparison," *Master Thesis*, King Fahd University of Petroleum and Minerals, June 1996.

[19] Parvis Kermani, Leonard Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, Vol. 3, No. 4, pp. 267-286, September 1979.

[20] Forde B. Nilsen, "Methods For Performance Evaluation of Wormhole-Switched Networks," *Doctoral Dissertation*, University of Oslo, July 1998.

[21] Po-Chi Hu, Leonard Kleinrock, "A Simple Host Deflection Scheme for High-Speed LANS Using Wormhole Routing," *In Proceedings of the 1996 International Conference on Network Protocols*, pp. 124-131, 1996.

[22] Mario Gerla, B. Kannan, Kwan Bruce, Palnati Prasath, Walton Simon, Leonardi Emilio and Neri Fabio, "Quality of Service Support in High-Speed, Wormhole Routing Networks," *In Proceedings of the 1996 International Conference on Network Protocols*, pp. 1-22, October 1996.

[23] Harish Sethu, Craig B. Stunkel, Robert F. Stucke, "IBM RS/6000 SP Interconnection Network Topologies for Large Systems," *In proceedings of the1998 International Conference on Parallel Processing*, pp. 620-627, 1998.

[24] José Duato, Pedro López, Federico Silla, Sudhakar Yalamanchili, "A High Performance Router Architecture For Interconnection Networks," *In Proceedings of the 1996 International Conference on Parallel Processing*, pp. 61-68, August 1996.

[25] José Duato, Pedro López, Sudhakar Yalamanchili, "Deadlock- and Livelock-Free Routing Protocols for Wave Switching," *In Proceedings of the 11th International Parallel Processing Symposium*, pp. 570-577, 1997.

[26] Kang G. Shin, Stuart W. Daniel, "Analysis and Implementation of Hybrid Switching," *IEEE Transactions on Computers*, Vol. 45, No. 6, pp. 684-692, June 1996.

[27] James Dolter, Stuart Daniel, Ashish Mehra, Jennifer Rexford, Wu-chang Feng, Kang Shin, "SPIDER: Flexible and Efficient Communication Support for Point-to-Point Distributed Systems," *In Proceedings of the International Conference on Distributed Computing Systems*, pp. 574-580, June 1994.

[28] William J. Dally, "Virtual-Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 2, pp. 194-205, March 1992.

[29] Wu-Chang Feng, Kang G. Shin, "Impact of Selection Functions on Routing Algorithm Performance in Multicomputer Networks," *In Proceedings of the 11th Annual Conference on Supercomputing*, pp. 1-8, July 1997.

[30] Timothy Mark Pinkston, Sugath Warnakulasuriya, "On Deadlocks in Interconnection Networks," *In Proceedings of the 24th International Symposium on Computer Architecture*, pp. 38-49, June1997.

[31] Sugath Warnakulasuriya, Timothy Mark Pinkston, "Characterization of Deadlocks in Interconnection Networks," *Proceedings of the 11th International Parallel Processing Symposium*, pp. 80-86, April 1997.

[32] Christopher J. Glass, Lionel M. Ni, "The Turn Model for Adaptive Routing," *Journal of the Association of Computing Machinery*, Vol. 41, No. 5, pp. 874-902, September 1994.

[33] William J. Dally, Charles L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, Vol. C-36, No. 5, pp. 547-553, May 1987.

[34] Andrew A. Chien, "A Cost and Speed Model for k-ary n-cube Wormhole Routers," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 2, pp. 150-162, February 1998.

[35] Andreas G. Nowatzyk, Michael C. Browne, Edmund J. Kelly, Michael Parkin, "S-Connect: from Networks of Workstations to Supercomputer Performance," *Proceedings of the 22nd International Symposium on Computer Architecture*, pp. 71-82, June 1995.

[36] Jae H. Kim, "Planar-Adaptive Routing (PAR): Low-Cost Adaptive Networks for Multiprocessors," *Master Thesis*, University of Illinois at Urbana-Champaign, 1993.

[37] Jose Duato, "A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Transactions on Parallel And Distributed Systems*, Vol. 6, No. 10, pp. 1055-1067, October 1995.

[38] Xiaola Lin, Philip K. McKinley, Lionel M. Ni, "The Message Flow Model for Routing in Wormhole-Routed Networks," *IEEE Transactions on Parallel And Distributed Systems*, Vol. 6, No. 7, pp. 755-760, July 1995.

[39] Jose Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Transactions on Parallel And Distributed Systems*, Vol. 4, No. 12, pp. 1320-1331, December 1993.

[40] P. López, J. M. Martínez, J. Duato, "A Very Efficient Distributed Deadlock Detection Mechanism for Wormhole Networks," *In Proceedings of the Fourth International Symposium on High-Performance Computer Architecture*, pp. 57-66, February 1998.

[41] Po-Chi Hu, Leonard Kleinrock, "A Dynamic Timeout Scheme for Wormhole Routing Networks," *In Proceedings of the IEEE International Conference on Communications*, pp. 1406-1410, June 1997.

[42] J. M. Martínez, P. López, J. Duato, T. M. Pinkston, "Software-Based Deadlock Recovery Technique for True Fully Adaptive Routing in Wormhole Networks," *In Proceedings of the 1997 International Conference on Parallel Processing*, pp. 182-189, August 1997.

[43] Jae H. Kim, Ziqiang Liu, Andrew A. Chien, "Compressionless Routing: A Framework for Adaptive and Fault-Tolerant Routing," *IEEE Transactions on Parallel And Distributed Systems*, Vol. 8, No. 3, pp. 229-244, March 1997.

[44] Po-Chi Hu, Leonard Kleinrock, "A Queueing Model for Wormhole Routing with Timeout," *In Proceedings of the 4th International Conference on Computer Communications and Networks*, pp. 584-593, September1995.

[45] Anan K. V., Timothy Mark Pinkston, "An Efficient, Fully Adaptive Deadlock Recovery Scheme: DISHA," *In Proceedings of the 22nd International Symposium on*

Computer Architecture, pp. 201-210, June 1995.

[46] Timothy Mark Pinkston, "Flexible and Efficient Routing Based on Progressive Deadlock Recovery," *IEEE Transactions on Computers*, Vol. 48, No. 7, pp. 649-669, July 1999.

[47] Charles L. Seitz, Wen-King Su, "A Family of Routing and Communication Chips Based on the Mosaic," *In Proceedings of the University of Washington Symposium on Integrated Systems*, pp. 1-18, 1993.

[48] Charles L. Seitz, Nanette J. Boden, Jakov Seizovic, Wen-King Su, "The Design of the Caltech Mosaic C Multicomputer," *In Proceedings of the University of Washington Symposium on Integrated Systems*, pp. 1-22, 1993.

[49] Andrew Chien, Jae H. Kim, "An Evaluation of Planar-Adaptive Routing (PAR)," *In Proceedings of the Symposium on Parallel and Distributed Processing*, pp. 470-478, 1992.

[50] Kazuhiro Aoyama, "Design Issues in Implementing an Adaptive Router," *Master Thesis*, University of Illinois at Urbana-Champaign, 1993.

[51] Christopher J. Glass, Lionel M. Ni, "Fault-Tolerant Wormhole Routing in Meshes without Virtual Channels," *IEEE Transactions on Parallel And Distributed Systems*, Vol. 7, No. 6, pp. 620-636, June 1996.

[52] Luis Gravano et al, "Adaptive Deadlock- and Livelock-Free Routing With All Minimal Paths in Torus Networks," *IEEE Transactions on Parallel And Distributed Systems*, Vol. 5, No. 12, pp. 1233-1251, December 1994.

[53] Swaminathan Ramany, Derek Eager, "The Interaction between Virtual Channel Flow Control and Adaptive Routing in Wormhole Networks," *ACM Online Digital Library //www.acm.org/dl/*, pp. 136-145, July 1994.

[54] Yen-Wen Lu et al, "A Comparison of Different Wormhole Routing Schemes," *In Proceedings of the 2nd International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 323-328, 1994.

[55] Anan K. V., Timothy Mark Pinkston, Jose Duato, "Generalized Theory for Deadlock-Free Adaptive Wormhole Routing and its Application to *Disha* Concurrent," *In Proceedings of the 1996 International Parallel Processing Symposium*, pp. 815-821, 1996.

[56] Younes M. Boura, Chita R. Das, "Performance Analysis of Buffering Schemes in Wormhole Routers," *IEEE Transactions on Computers*, Vol. 46, No. 6, pp. 687-694, June 1997.

[57] Jong Kim, Chita R. Das, "Hypercube Communication Delay with Wormhole Routing," *IEEE Transactions on Computers*, Vol. 43, No. 7, pp. 806-814, July 1994.

[58] Jennifer Rexford, Wu-chang Feng, James Dolter, Kang G. Shin, "PP-MESS-SIM: A Flexible and Extensible Simulator for Evaluating Multicomputer Networks," *IEEE Transactions on Parallel And Distributed Systems*, Vol. 8, No. 1, pp. 25-40, January 1997.

[59] Raj Jain, "The Art of Computer Systems Performance Analysis; Techniques for Experimental Design, Measurement, Simulation, and Modeling," *John Wiley & Sons Inc.*, 1991.

[60] P. López, J. M. Martínez, J. Duato, "Impact of Buffer Size on the Efficiency of Deadlock Detection," *In Proceedings of the Fifth International Symposium on High-Performance Computer Architecture*, pp. 315-318, January 1999.

[61] Zaki H. Al-Awwami, M. S. Obaidat, M. Al-Mulhem, "A New Deadlock Recovery Mechanism for Fully Adaptive Routing Algorithms", *In Proceedings of the 19th IEEE International Performance, Computing, and Communications Conference*, pp. 132-138, February 2000.

[62] Smaragda Konstantinidou, Lawrence Snyder, "The Chaos Router," *IEEE Transactions on Computers*, Vol. 43, No. 12, pp. 1386-1397, December 1994.

[63] Kevin Bolding, "Chaotic Routing – Design and Implementation of an Adaptive Multicomputer Network Router," *Doctoral Dissertation*, University of Washington,

1993.

[64] Yungho Choi, Timothy Mark Pinkston, "Crossbar Analysis for Optimal Deadlock Recovery Router Architecture," *In Proceedings of the 11th International Parallel Processing Symposium*, pp. 583-588, April 1997.

[65] Emilio Leonardi, Fabio Neri, Mario Gerla, Prasasth Palnati, "Congestion Control in Asynchronous, High-Speed Wormhole Routing Networks," *IEEE Communications*, pp. 58-69, November 1996.

[66] Fabrizio Petrini, Marco Vanneschi, "Minimal Adaptive Routing with Limited Injection on Toroidal k-ary n-cubes", *In Proceedings of the 1996 Supercomputing Conference*, Article No. 23, 1996.

[67] Pedro Lopez, Jose Duato, "Deadlock-Free Adaptive Routing Algorithms for the 3D-Torus: Limitations and Solutions" *In Proceedings of the Parallel Architectures and Languages Europe 93*, pp. 684-687, June 1993.

[68] Fabrizio Petrini, Jose Duato, Pedro Lopez, Juan-Miguel Martinez, "LIFE: a Limited Injection, Fully adaptivE, Recovery-Based Routing Algorithm", *In Proceedings of the 4th International Conference on High Performance Computing*, pp. 589-595, December 1997.

[69] P. Lopez, J. M. Martinez, J. Duato, F. Petrini "On the Reduction of Deadlock Frequency by Limiting Message Injection in Wormhole Networks", *In Proceedings of the Parallel Computer Routing and Communications Workshop*, pp. 295-307, June 1997.

[70] P. Lopez, J. M. Martinez, J. Duato, "DRIL: Dynamically Reduced Message Injection Limitation Mechanism for Wormhole Networks", *In Proceedings of the 1998 International Conference on Parallel Processing*, pp. 535-542, August 1998.

[71] M. S. Obaidat, Zaki H. Al-Awwami, M. Al-Mulhem, "CLIC: A New Injection Limitation Mechanism to Improve the Performance of Fully Adaptive Routing in Wormhole Networks", *In Proceedings of the 2001 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, pp. 105-114, July 2001.

Vita

- Zaki Al-Awwami

- Born on September 27, 1964 in Qatif, Saudi Arabia.

- Received Bachelor of Science degree in Computer Engineering from the College of Engineering at the University Of the Pacific (UOP) in Stockton, California, USA, 1987.

- Working as a Senior Systems Analyst for Saudi Aramco from 1987 until the current time.

- Obtained a Master Degree in Computer Science from the Information and Computer Science Department at King Fahd University of Petroleum and Minerals in Dhahran, Saudi Arabia, October 2001.

- List of Publications:
  - IPCCC 2000 (International Performance Computing and Communications Conference), Phoenix, AZ, USA, February 20-22, 2000.
  - The SI Simulation Journal (Transactions of SCS-Part B), 2001.
  - SPECTS 2001 (International Symposium on Performance Evaluation of Computer and Telecommunications Systems), Orlando, Florida, USA, July 15-19, 2001.
  - ICCNMC 2001 (International Conference on Computer Networks and Mobile Computing), Beijing, China, October 16-19, 2001.
  - Special issue of the Computer Communications Journal, 2002.